Fall 2010

Structure and content semantic similarity detection of eXtensible markup language documents using keys

Waraporn Viyanon

Department: Computer Science

STRUCTURE AND CONTENT SEMANTIC SIMILARITY DETECTION OF EXTENSIBLE MARKUP LANGUAGE DOCUMENTS USING KEYS

by

WARAPORN VIYANON

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2010

Approved by

Dr. Sanjay Madria, Advisor
Dr. Fikret Ercal
Dr. Jennifer Leopold
Dr. Chaman Sabharwal
Dr. Vincent Yu

PUBLICATION DISSERTATION OPTION

This dissertation consists of four articles prepared in the style required by the journals or conference proceedings in which they were published:

Pages 23 to 37, "XML Data Integration Based on Content and Structure Similarity Using Keys", were published in the 16th International Conference on Cooperative Information Systems (CooPIS 2008), Monterrey, Mexico.

Pages 38 to 64, "A System for Detecting XML Similarity in Content and Structure Using Relational Database", were published in the Proceedings of 18th ACM International Conference on Information and Knowledge Management (ACM CIKM 2009), Hong Kong, China.

Pages 65 to 89, "XML-SIM: Structure and Content Semantic Similarity Detection Using Keys", were published in the 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2009), Vilamoura, Algarve-Portugal.

Pages 90 to 113, "XML-SIM-CHANGE: Structure and Content Semantic Similarity Detection among XML Document Versions", were published in the 9th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2010), Crete, Greece.

Some parts of the literature review have been included as a part of a book chapter. This book chapter has been submitted and under revision for the book named XML Data Mining: Models, Methods, and Applications.

ABSTRACT

XML (eXtensible Mark-up Language) has become the fundamental standard for efficient data management and exchange. Due to the widespread use of XML for describing and exchanging data on the web, XML-based comparison is central issues in database management and information retrieval. In fact, although many heterogeneous XML sources have similar content, they may be described using different tag names and structures.

This work proposes a series of algorithms for detection of structural and content changes among XML data. The first is an algorithm called XDoI (XML Data Integration Based on Content and Structure Similarity Using Keys) that clusters XML documents into subtrees using leaf-node parents as clustering points. This algorithm matches subtrees using the key concept and compares unmatched subtrees for similarities in both content and structure. The experimental results show that this approach finds much more accurate matches with or without the presence of keys in the subtrees. A second algorithm proposed here is called XDI-CSSK (a system for detecting xml similarity in content and structure using relational database); it eliminates unnecessary clustering points using instance statistics and a taxonomic analyzer. As the number of subtrees to be compared is reduced, the overall execution time is reduced dramatically. Semantic similarity plays a crucial role in precise computational similarity measures. A third algorithm, called XML-SIM (structure and content semantic similarity detection using keys) is based on previous work to detect XML semantic similarity based on structure and content. This algorithm is an improvement over XDI-CSSK and XDoI in that it determines content similarity based on semantic structural similarity. In an experimental

evaluation, it outperformed previous approaches in terms of both execution time and false positive rates.

Information changes periodically; therefore, it is important to be able to detect changes among different versions of an XML document and use that information to identify semantic similarities. Finally, this work introduces an approach to detect XML similarity and thus to join XML document versions using a change detection mechanism. In this approach, subtree keys still play an important role in order to avoid unnecessary subtree comparisons within multiple versions of the same document. Real data sets from bibliographic domains demonstrate the effectiveness of all these algorithms.

ACKNOWLEDGMENTS

I would like to express my deep and sincere gratitude to my advisor, Dr. Sanjay Kumar Madria, for his support, guidance, and motivation to finish this dissertation. His knowledge, skills, and advice have been of great value to me. It has been a wonderful experience to work under him.

I would also like to thank the members of my advisory committee, Dr. Fikret Ercal, Dr. Jennifer L. Leopold, Dr. Chaman Sabharwal, and Dr. Wen-Bin (Vincent) Yu, for their valuable feedback and guidance, which helped improve my research.

I am also grateful to other faculty members, graduate students, colleagues, and friends at Missouri S&T: Dr. Ali Hurson, Clayton Price, Dr. Ralph W. Wilkerson, Dr. Xiaoqing (Frank) Liu, Cyriac Kandoth, Leong Lee, Dylan McDonald, Roy Cabaniss, Vimal Kumar, Rhonda Grayson, and Dawn Davis. They and many others have contributed to my work in many ways and made my time in Rolla enjoyable.

I wish to express particular appreciation to Chalee Intrarakhanchit for his tips on Oracle troubleshooting.

Finally, none of this would have been possible without the love, support, understanding, and encouragement of my parents, brothers, and sister.

TABLE OF CONTENTS

PAPER

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

XML (eXtensible Mark-up Language) has become increasingly important as the fundamental standard for efficient data management and exchange [25]. Information designed to be broadcast over the Internet is represented in XML to ensure its interoperability. The use of XML covers a wide variety of applications ranging from data storage and representation to database information interchange, data filtering, and web services interaction.

As the use of XML to describe and exchange data on the web has grown, comparison of XML documents has become central issues in database management and information retrieval. Applications of XML similarity analysis are numerous and include: (i) data integration, (ii) version control, (iii) classification and clustering of XML documents, and (iv) XML query systems [60].

Although heterogeneous XML sources may have similar content, they may be described using different tag names and structures. Examples include the bibliography data sources; DBLP [71] and Sigmod Record [2]. Integration of similar XML documents from different data sources benefits users, giving them access to more complete and useful information and query systems to retrieve information from a single integrated source instead of various sources.

XML is a structured format, which means that the arrangement, organization, and expression of data in a document can be defined exactly. The structure of XML documents organized in a hierarchical manner shows how elements stand in relation to each other. The content of XML documents holds the meaning. Since XML documents encode both structure and data, accurate measurement of similarities among the documents and integration of documents demand consideration of similarities in both the structure and content of XML documents.

There is much research to be done on the structural similarities of XML documents. Tree edit distance (TED) [16] is one method to estimate similarities among hierarchically structured data. This technique determines which edit operations to transform one tree into another have the lowest cost. Several projects have relied on TED

between corresponding trees of XML documents; however, evaluating TED is computationally expensive and difficult to scale up to large collections [16, 59, 62].

Precise measurement of similarities between XML documents must take into account the semantics of those documents. Some work has considered both content and structural similarities, such as LAX, SLAX, and composite SVM kernels [41, 42, 28].

The techniques presented here to detect degrees of similarities among XML documents being by clustering XML documents into smaller subtrees, each considered an individual object, using leaf-node parents. The clustered subtrees are filtered using a taxonomic analyzer and the instance statistics concept. The taxonomic analyzer is a method of determining how close the meanings of element names are and transforming them into a single category. For example, an XML document may contain an element name “Pages,” which has two sub-elements, “initPage” and “endPage,” as descendants. These three element names can be categorized in the same group using the taxonomic analyzer. The instance statistic [68] is employed to determine the relationships among the element names. Subtrees are considered proper if they show a one-to-one relationship between XML elements. Once the subtrees are filtered, the concept of key is exploited. Keys play an essential part in the database design [13] techniques presented here, permitting matching of subtrees between documents. In addition, in order to compare documents, this work uses the results of matching with keys to identify irrelevant and therefore inappropriate leaf-node parents. The results of experiments show that overall computation is improved without compromising quality.

2. REVIEW OF LITERATURE

2.1. PRELIMINARIES

2.1.1. XML Data Model. An XML document can be represented as a tree. Each node[1] of the tree corresponds to an XML element, which is written with an opening and closing tag. Each edge represents parent/child relationship between elements in the XML file. The leaf nodes of the tree represent data values according to their relationship. A path from the root element to a leaf node element is called a path expression[2] or path signature. In Figure 2.1, '/book/title/' is the path signature for the data "XML in Use."

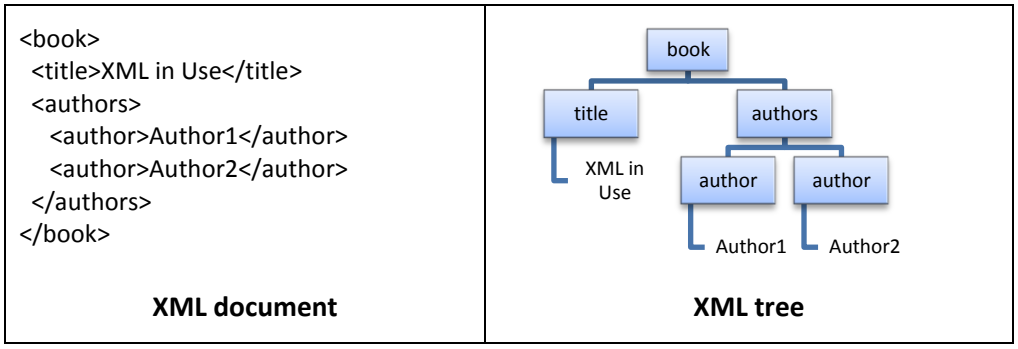


Figure 2.1. XML data model

2.1.2. Views of XML. As Bourret [12] pointed out, XML documents can be classified as having either: (1) a document-centric (text-centric) view or (2) a data-centric view.

Data-centric documents are used to transport data. As such, they are highly structured data marked up with XML tags. Most data-centric XML documents are generated from structured sources such as RDBMS. The data-centric view emphasizes on XML structure since the meaning of a data-centric XML document depends only on the

[1] The terms 'node' and 'element' are used interchangeably in this paper.

[2] The term 'path expression' and 'path signature' are used interchangeably in this paper.

structured data represented inside it. It is usually used to exchange data in a structured form.

Document-centric documents focus on application-relevant objects. They are loosely structured documents marked-up with XML tags, and their meaning depends on the document as a whole. Their structure is more irregular, and their data are heterogeneous. Such documents might not even have a document-type declaration (DTD) or XML schema. For this view, text is a higher priority than structure. Figure 2.2 shows examples of both document-centric and data-centric documents.

```
<FlightInfo>
  <Airline>ABC Airways</Airline> provides
<Count>three</Count>
  non-stop flights daily from <Origin>Dallas</Origin> to
  <Destination>Fort Worth</Destination>. Departure
times are
  <Departure>09:15</Departure>,
<Departure>11:15</Departure>,
  and <Departure>13:15</Departure>. Arrival times are
minutes later.
</FlightInfo>
```

(a)

```
<Flights>
  <Airline>ABC Airways</Airline>
  <Origin>Dallas</Origin>
  <Destination>Fort Worth</Destination>
  <Flight>
    <Departure>09:15</Departure>
    <Arrival>09:16</Arrival>
  </Flight>
  <Flight>
    <Departure>11:15</Departure>
    <Arrival>11:16</Arrival>
  </Flight>
  <Flight>
    <Departure>13:15</Departure>
    <Arrival>13:16</Arrival>
  </Flight>
</Flights>
```

(b)

Figure 2.2. Two types of XML documents: (a) document-centric (b) data-centric

2.1.3. Benefits of XML. Daly [21] outlines the benefits of XML, explaining why it is an effective solution for the design of a wide range of applications.

(1) XML is simple. It codes information coded in a format that is easy for humans to read and understand, and easy for applications to process.

(2) XML is extensibility. It has no fixed set of fields. New fields can be created as needed.

(3) XML is self-describing. In traditional databases, data records require schemas set up by the database administrator. Because they contain meta-data in the form of fields and attributes, XML documents can be stored without such definitions. XML also provides a basis for author identification and versioning at the element level. Any XML field can possess an unlimited number of attributes, such as author or version.

(4) XML is a World Wide Web Consortium (W3C) standard endorsed by software industry market leaders.

(5) XML supports multilingual documents and Unicode; it is appropriate for the international applications.

(6) XML facilitates the comparison and aggregation of data. The tree structure of XML documents allows documents to be compared and aggregated efficiently, element-by-element.

(7) XML can embed multiple data types. XML documents can contain any possible data type - from multimedia data (image, sound, and video) to active components (Java applets, ActiveX).

(8) XML can embed existing data. Mapping existing data structures like file systems or relational databases to XML is simple. XML supports multiple data formats and can cover all existing data structures.

2.2. CHALLENGES IN DETECTION OF XML SIMILARITIES

2.2.1. Performance. Generally, XML similarity detection relies on a large number of comparisons among subtrees in DOM (document object model) trees. To identify simple and reasonable properties of the match and merge functions, efficient processing and optimal algorithms are needed.

2.2.2. Scalability. Most XML documents, such as protein sequence data sets, particularly in XML data integration, are large. Efficient matching and merging functions require that the data sets are loaded, which may not allow them to fit into the main memory to fetch and write results to disk as efficiently as possible. Secondary storage may be required in this case. One possible solution is to store XML documents in a

relational database, which requires a mapping technique to maintain the structure and content of the XML documents.

2.2.3. Similarity Measures. Similarity measures play a key role in analyzing XML similarity. Selecting or building a similarity approach is important for accuracy. Many approaches to measuring similarity must be compared and improved for efficiency.

2.3. MEASUREMENT OF SIMILARITY AND DISTANCE

The concept of similarity has been the subject of much research in the fields of computer science, psychology, artificial intelligence, and linguistics. Typically, such studies focus on the similarity between vectors, strings, trees, or objects. The input data comes from two documents, X and Y. Each similarity measure considered is presented as a function of X and Y, $sim(\mathrm{X}, Y)$, indicating the degree of similarity between documents X and Y. The similarity value is a number between 0 and 1. The similarity is minimized only if the two sets share nothing in common, and it is maximized only if the two sets are identical. In order to decide which pair is similar, a threshold is defined. The similarity measures described below normalize the overlap and distance in various ways.

2.3.1. Vector-Space Data Similarity

2.3.1.1 L1 Norm. L1 norm is a means of calculating the distance between two points by calculating the sum of the differences:

$$L_1(X, Y) = \sum_i |x_i - y_i|. \tag{1}$$

2.3.1.2 Euclidean Distance. Euclidean distance (also called L2 norm) is a means of determining the distance between two points in mathematics. It can be applied to find the similarity between two documents:

$$L_2(X, Y) = \sqrt{\sum_i (x_i - y_i)^2}. \tag{2}$$

2.3.1.3 Dice Similarity. This is a similarity measure defined as twice the shared information (intersection) over the combined set (union):

$$DiceSim(X,Y) = \frac{2|X \cap Y|}{|X \cup Y|}. \quad (3)$$

2.3.1.4 Jaccard Similarity. This similarity metric is similar to dice similarity. The Jaccard coefficient measures similarity between two sets; it is defined as the size of the intersection divided by the size of the union of the given sets:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}. \quad (4)$$

2.3.1.5 Cosine Similarity. The cosine of the angle between two vectors x and y, the cosine similarity, θ, is represented using a dot product and magnitude as

$$cosine(X,Y) = \frac{X \cdot Y}{|X||Y|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x^2} \cdot \sqrt{\sum_i y^2}}. \quad (5)$$

As the angle between the vectors narrows two vectors draw closer to one another. The measure can be transformed in set notation:

$$CosineSim(X,Y) = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}}. \quad (6)$$

This solution is easy to implement, but it suffers from several limitations. The structures of XML documents are often ignored in the comparison process. If the dimensions of the vector become larger, the value of some features becomes unavailable. In addition, the order in which the terms appear in the document is lost in the vector space representation.

2.3.1.6 Overlap Similarity. This measure computes the overlap between two sets defined as

$$overlap(x,y) = |x \cap y|. \quad (7)$$

2.3.1.7 Term Frequency-Inverse Document Frequency Weights. The vector space model was proposed by Salton [75]. The term Tf-idf weight is an abbreviation of *term frequency-inverse document frequency*. This approach defines a weight for each document term. Terms are typically single words, keywords, or longer phases. The more

common a term, the lower its weight. This weighting prevents a bias toward longer documents and gives a measure of the importance of the term t_i within the particular document d_j**.** The weight vector can be defined as

$$w_{i,j} = tf_{i,j} \cdot log \frac{|D|}{|\{d: t_i \in d\}|} \tag{8}$$

where $tf_{i,j}$ is the number of occurrences. The term (t_i) in document d_j, divided by the sum of the number of occurrences of all terms in document. The term $log \frac{|D|}{|\{d:t_i \in d\}|}$ is the inverse document frequency, $|D|$ is the number of documents in the document sets, and $|\{d: t_i \in d\}|$ is the number of documents in which the term t_i appears.

2.3.2. String Similarity

2.3.2.1 String Matching. String matching is important in the domain of text processing. It involves identifying a place where one or several strings are found within a text. String matching algorithms generally scan the text with the help of a window. They first align the left ends of the window and the text, then compare the characters of the window (called patterns). After a whole match of the pattern, or after a mismatch, they shift the window to the right. The same procedure is repeated until the right end of the window goes beyond the right end of the text. This mechanism is called the *sliding window mechanism*.

2.3.2.2 Longest Common Subsequence. Longest common subsequence (LCS) [44] determines the longest subsequences that can be obtained by deleting zero or more symbols from each of two given sequences. This is an NP-hard problem. Given two sequences be defined as $X = (x_1, x_2, \dots, x_m)$ and $Y = (y_1, y_2, \dots, y_n)$, LCS function is defined as

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & if\ i = 0\ or\ j = 0 \\ (LCS(X_{i-1}, Y_{j-1}), x_i) & if\ x_i = y_j \\ longest\left(LCS(X_i Y_{j-1}), LCS(X_{i-1}, Y_j)\right) & if\ x_i \neq y_j \end{cases}. \tag{9}$$

This method is suitable for biological data integration applications. Exact string matching often fails to associate a name with its biological concept (i.e., ID or accession number in the database) due to seemingly small differences between names. Soft string matching could permit identification of relevant information by considering the similarity

between names. However, the accuracy of soft matching depends heavily on the similarity measure used.

2.4. SEMANTIC SIMILARITY MEASURES

Semantic similarity rests on the concept that a set of documents or terms within a term list can be assigned a metric based on the relatedness of their semantic content. Semantic similarity methods [32, 38, 43, 49, and 53] have been introduced to capture the meaning of words. Generally, these methods can be categorized into two main groups: edge-counting-based methods [50] and information-corpus-based methods.

Semantic similarity requires a lexical database. One large lexical database used in many natural language processing (NLP) applications is WordNet. It includes most English nouns, verbs, adjectives, and adverbs.

WordNet [26] is organized by meaning: Words in close proximity are semantically similar. It is classified in terms of *synsets*, or unordered sets of roughly synonymous words or multiword phases. Each synset expresses a distinct meaning or concept. Figure 2.3 (a) shows some examples of synsets. Each set contains synonymous words and their meaning.

A taxonomy is represented in a hierarchal form consisting of nodes and edges. Each node represents a synset, and each edge indicates a semantic relationship between synsets. These relationships could be hyperonyms, hyponyms, holonyms, meronyms, coordinate terms, troponyms, or entailments. Figure 2.3 (b) depicts the relationships among synsets.

The Cambridge/Acquilex lexical database system is a computer system that provides flexible access to machine-readable dictionaries. It supports user retrieval of subsets of entries from one or more dictionaries, including the Longman Dictionary, the MRC Psycholinguistic Database, the Van Dale Dutch monolingual, and bilingual dictionaries. This system is a hierarchical collection of attributes and associated values that can be described in terms of syntax (syn) and semantic (sem). A query can be mapped onto a collection of indices to determine which synsets are most discriminating.

Noun

{pipe, tobacco pipe} (a tube with a small bowl at one end; used for smoking tobacco)

{pipe, pipage, piping} (a long tube made of metal or plastic that is used to carry water, oil, gas, etc.)

{pipe, tube} (a hollow cylindrical shape)

{pipe} (a tubular wind instrument)

{organ pipe, pipe, pipework} (the flues and stops on a pipe organ)

Verb

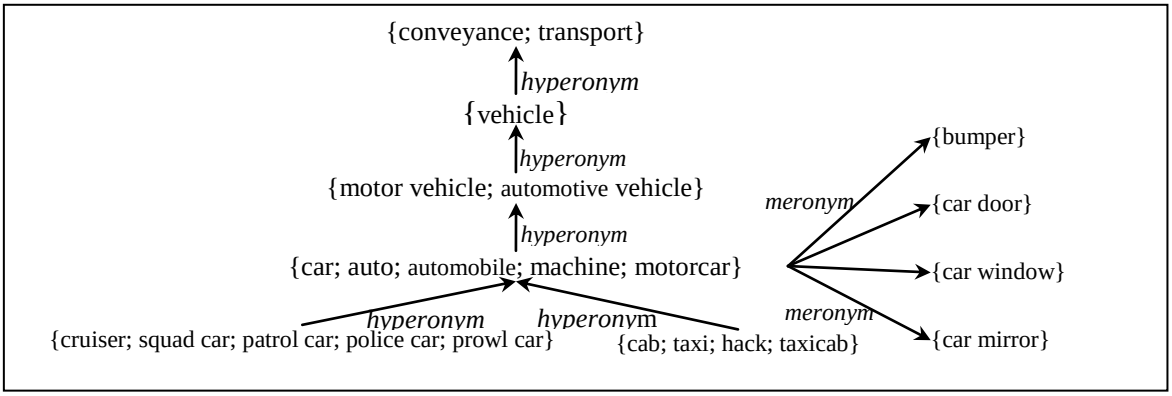
{shriek, shrill, pipe up, pipe} (utter a shrill cry)

{pipe} (transport by pipeline) "pipe oil, water, and gas into the desert"

{pipe} (play on a pipe) "pipe a tune"

{pipe} (trim with piping) "pipe the skirt"

(a)



(b)

Figure 2.3 Example of synsets: (a) WordNet's synsets (b) part of WordNet

2.4.1. Node and Edge Metrics

2.4.1.1 Edge-counting Based Metric. The edge-based approach is a simple and intuitive way of evaluating semantic similarity in a taxonomy. This approach estimates the distance between nodes corresponding to the concepts being compared. This geometric distance can be measured. Rada et al. [50] showed that the simplest means of determining the distance between two concept nodes, A and B, is identifying the shortest path that links A and B, or the minimum number of edges that separate A and B.

Jiang and Conrath [32] have noted that the distance between any two adjacent nodes is not necessarily equal; therefore, this approach is not sensitive to the problem of

varying link distances. Edge weight can be considered in order to solve this problem. It is related to the number of children, the depth of a node in the hierarchy, the type of link (such as the is-a, part-of, or substance-of links), the network density, and the strength of an edge link.

2.4.1.2 Path Length Metric. Leacock and Chodoro's measure of similarity [36] relies on the length $\boldsymbol{len(c_1, c_2)}$ of the shortest path between two synsets:

$$Sim_{LC}(c_1, c_2) = -\log \frac{\text{len}(c_1, c_2)}{2\text{D}} \tag{10}$$

where D is the overall depth of the taxonomy. This measure is limited by its attention to IS-A links and to the scale of the path length, or the depth of the taxonomy.

2.4.1.3 Node Depth Metric. This method measures the depth of two concepts in a taxonomy and the depth of the least common subsume (LCS). It then combines these properties into a similarity score:

$$Sim_{Wu-Palmer}(c_1, c_2) = \frac{2 \times depth(LCS(c_1, c_2))}{depth(c_1) + depth(c_2)} \tag{11}$$

where LCS is the lowest common subsume and $depth\ (c)$ is the depth of node c in the hierarchy.

2.4.2. Information Corpus-based Methods. Several research groups [53, 32, 43] have proposed information-content (IC) based measures of semantic similarity between terms. These measures were designed mainly for WordNet.

2.4.2.1 Resnik's Measure. Resnik's measure calculates the semantic similarity between two terms $[t_1, t_2]$ in a given ontology (e.g., WordNet) as the information content (IC) of the least common ancestor (LCA) of $t_1 and\ t_2$. The IC of a term t can be quantified in terms of the probability $(P(t))$ of its occurrence. The probability assigned to a term is defined as its relative frequency of occurrence:

$$Sim_{Resnik's} = IC(c) = -\log P(c) = -\log P(LCS(c_1, c_2)) \tag{12}$$

where P(c) is the probability that a randomly selected word in a corpus is an instance of concept c. This can also be written as

$$P(c) = \frac{\sum_{w \in words(c)} count(w)}{N} \quad (13)$$

where words(c) is the set of words subsumed by concept c, and N is the total number of words in the corpus.

2.4.2.2 Jiang and Conrath Distance. Intuitively, the more differences between A and B, the less similar they are. Conversely, the more A and B have in common, the more similar they are. Jiang and Conrath distance uses the notion of information content and the probability of encountering an instance of a child-synset given an instance of an LCS. Thus, the information content of the two nodes, as well as that of their most specific subsume, plays a part:

$$dist_{JC}(c_1, c_2) = 2 \times \log P\big(LCS(c_1, c_2)\big) - (\log P(c_1) + \log P(c_2)). \quad (14)$$

Note that the output for this equation is distance, the inverse of similarity.

2.4.2.3 Lin's Measure. Lin's similarity measure follows from his theory of similarity between arbitrary objects. It uses the same elements as Jiang and Conrath distance, but in a different fashion:

$$Sim_{Lin}(c_1, c_2) = \frac{2 \times \log P(\mathrm{LCS}(c_1, c_2))}{\log\ \mathrm{P}(c_1) + \log \mathrm{P}(c_2)}. \quad (15)$$

2.4.3. Hybrid Similarity. Semantic similarity plays an important role in finding the similarity in meaning or semantic content. Syntactic similarity measures have performed strongly with resources containing large amounts of text, but they cannot appropriately cope with syntactic and semantic heterogeneity and ambiguity if the semantics of the terms are not explicitly available. The hybrid similarity measure combines both semantic and syntactic similarity measures to detect the similarity among documents. This approach can be incorporated using average, maximum, additive, or weighted sum functions. The average, maximum and additive functions are simple. The weighted sum seems to work best, but it requires that the domain experts obtain the weights.

2.5. XML SIMILARITY

Much work has addressed XML similarity. Similarity can be computed at different layers of abstraction: at the data layer (i.e., similarity between data), at the type layer (i.e., similarity between types, also referred to as schema, models, or structures, depending on the application domain), or between the two layers (i.e., similarity between data and types). XML similarity can be categorized as either of two approaches: (1) structural similarity or (2) content and structural similarity. The XML documents thus compared are data-centric documents.

2.5.1. Structure Oriented Similarity. **S**tructural similarity focuses mainly on document classification or schema mapping in order to generate a global schema. A global schema is generated based on a formal merge ontology as a basis for integration and to resolve heterogeneity problems during integration. David Buttler [14] summarized three approaches to structural similarity: (1) tag similarity, (2) tree edit distance (TED), and (3) Fourier transform similarity.

2.5.1.1 Tag Similarity. This is the simplest way to measure the structural similarity of documents. It measures how close element names from two XML documents are. Documents that use similar element names are likely to have similar schema. This measure evaluates the number of intersected elements from the compared documents and it is divided by the union. In addition, the overlap can be calculated by applying a taxonomy to observe how similar element names are; however, this approach is not suitable for several reasons. One critical problem is that documents conforming to the same schema may have only a limited number of element names; one document may contain a large number of a particular element name, whereas the other may contain relatively few occurrences of the tag. In addition, tag similarity completely ignores the structure of documents, thus yielding low clustering quality.

2.5.1.2 Tree Edit Distance. Because XML documents can be represented in tree form, one popular technique to determine similarities between them is to determine the edit operations that can transform one tree into another with minimum cost. Edit operations can be classified in two groups: atomic edit operations and complex edit operations. An atomic edit operation can be either the deletion of an inner or leaf node, the insertion of an inner or leaf node, or the update of one node by another node. A

complex edit operation is the insertion, deletion, or update of a whole subtree. Tai [59] introduced the first nonexponential algorithm that has complexity of $O(|T_1| \times |T_2| \times depth(T_1)^2 \times depth(T_2)^2)$ when finding the minimum edit distance between trees T_1 and T_2. Here, $|T_1|$ and $|T_2|$ denote the number of nodes in T_1 and T_2 respectively; $depth(T_1)$ and $depth(T_2)$ are the depths of the trees.

Figure 2.4 shows an example of TED calculation of the similarity between trees T_1 and T_2. If nodes b, c ,and d are inserted in sequence, T_1 can transform T_2. The distance $Dist(T_1, T_2)$ between T_1 and T_2 is 3.

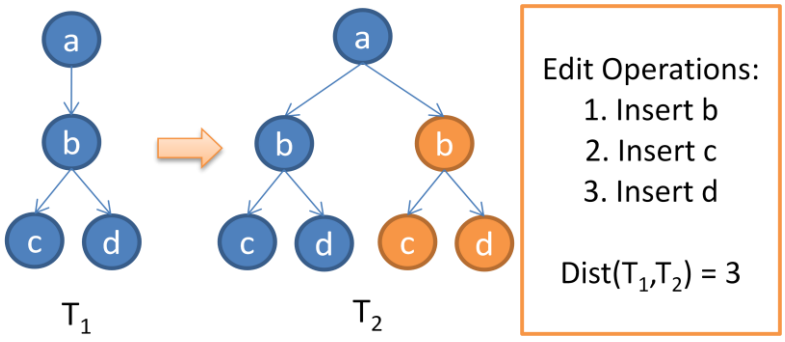


Figure 2.4. Atomic tree edit distance calculation

Previously, edit operations (insertion, deletion, substitution) have been allowed on single nodes only. If the distance between trees is computed by applying atomic edit operations as in Figure 2.5, the distance between T_A and T_B is equal to as calculated from the cost of inserting node h, b, c, d, and h. This cost is equal to the distance between T_A and T_C, the cost of inserting h, e, f, g, and h. In other words, T_B and T_C are the same distance from T_A. Obviously, T_B is more similar to T_A based on subtree structural commonalities (the complex tree edit operations) marked as circles in the XML tree comparisons in Figure 2.5.

Chawathe's approach [16] considers the insertion and deletion operations at the leaf-node level and allows replacement of node labels anywhere in the tree but, disregards the move operation. The overall complexity of Chawathe's algorithm is

expressed as $O(N^2)$ where N is the maximum number of nodes in the trees being compared. This method is computationally expensive and has a prohibitively high run time; therefore, it is not practical for similarity matching over large XML data repositories.

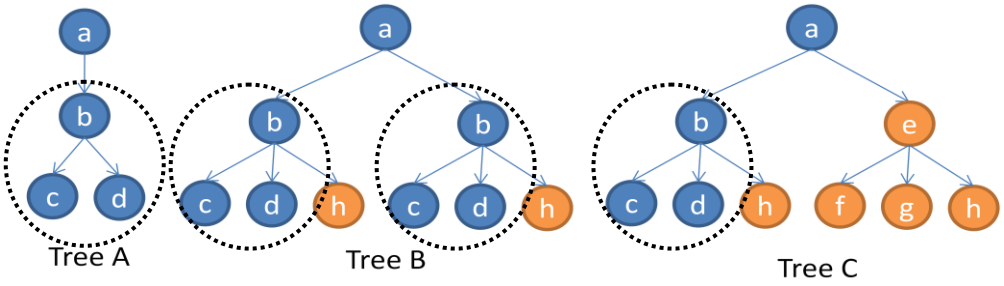


Figure 2.5. XML trees

Shasha and Zhang [58] propose a TED metric that permits the addition and deletion of single nodes anywhere in the tree, not just at the leaves. However, entire subtrees cannot be inserted or deleted in one step. The complexity of this approach is expressed as $O(|T_1||T_2|depth(T_1)depth(T_2))$.

Nierman and Jagadish [48] emphasize the identification of subtree structural similarities. Their edit operations are similar to Chawathe's, but they add two new operations: insert tree and delete tree. To determine subtree similarities, they introduce containment in the relationship between trees or subtrees. A tree T_1 is said to be contained in a tree T_2 if all nodes of T_1 occur in T_2 with the same parent/child edge relationship and node order. The overall complexity of this algorithm is expressed as $O(N^2)$. This approach proved more accurate in detecting XML structural similarities than those of either Chawathe or Shasha.

2.5.1.3 Fourier Transform Similarity. Essentially, Fourier transform similarity [51] removes all the information from a document except for its start and end tags, leaving only its skeleton, which represents its structure. The structure is then converted into a sequence of numbers, which is viewed as a time series, and a Fourier transform is

applied to convert the data into a set of frequencies. Finally, the distance between two documents is computed by calculating the difference in the magnitudes of the two signals. Buttler [14] proved that this algorithm is the least accurate of all approximation algorithms, and performs poorly because Fourier transform does not discriminate sufficiently between very similar documents.

2.5.1.4 Edge Matching. Lian et al. [39] represent XML document structures as directed graphs called s-graphs, and define a distance metric that captures the number of edges common to the graph representations of two XML documents:

$$Dist(G_1, G_2) = \frac{1 - |Edges(G_1) \cap Edges(G_2)|}{Max\{Edges(G_1),\ Edges(G_2)\}}. \quad (16)$$

This metric is more effective than others based on TED, in separating documents that are structurally different. It can be applied not only to tree-structured documents but also to document collections of arbitrary (graph) structure.

2.5.1.5 Path Similarity. Path expressions of XML documents can be used to find the similarity among these documents by measuring the similarity of paths between them [52]. A path is defined as a list of connected nodes starting at the root and terminating at a leaf node. Path similarity can be measured in several different ways: binary (where a path is either equivalent or not), partial (where the number of comparable nodes in each path is determined), or weighted (where the nodes are weighted according to their distance from the root). Rafiei, Moise and Sun [52] define two XML documents as similar if they share a large fraction of the paths in their path sets. The path set includes all root paths (from the root to leaf nodes) and all possible subpaths. The time complexity in terms of the number of string comparisons is expressed as $O(nl^2)$, where n is the number of root paths and $\boldsymbol{l}$ is the length of each path. Buttler [14] shows that the path similarity method provides fairly accurate results compared to TED.

2.5.1.6 XML/DTD Similarity. Structural similarity can be detected by comparing document type definitions (DTDs) with XML documents. Bertino, Guerrini, and Mesiti [8] proposed a matching algorithm for measuring the structural similarity between an XML document and a DTD. By comparing the document structure with that required by the DTD, the matching algorithm is able to identify commonalities and

differences. Differences can be the occurrence of extra elements beyond those required by the DTD, or the absence of required elements. The degree of similarity can be evaluated based on the element's properties, such as level or weight. Elements at higher levels are considered more relevant than those at lower levels. The authors state that their approach is of exponential complexity.

2.5.2. Similarity of XML Structure and Content. In the context of XML classification and clustering, structural similarity seems to be sufficient to distinguish or classify XML documents. However, in the context of XML data integration, not only the structure of XML documents must be considered, but also their content in order to determine whether XML documents have similar content to integrate.

2.5.2.1 Subtree Similarity. To integrate XML documents, many methods begin by with identifying objects in a data source that may represent real-world objects by clustering them into small fragments or subtrees. This method is called entity resolution (ER), also known as duplication or record linkage. A well clustered subtree should meet with the following requirements: (1) Each subtree represents one independent item, (2) each independent item is clustered into one subtree, and (3) the leaf nodes belonging to that item should be included in the subtree.

Liang and Yokota [41] provide an approach entitled leaf-clustering based approximate XML join algorithm (LAX). Their method consists of two main steps: (1) fragmenting XML documents into subtrees and (2) computing similarity. First, LAX divides XML trees into subtrees by considering a clustering point from the height (distance from the furthest child) and the number of link branches of XML trees. A link branch is a link between two candidate elements that have at least two children, or the distance of which to its furthest child is at least three. The subtree can be generated by deleting the link branch below the clustering point. The clustering point is calculated from the maximum weighting factor of the multiplication between the height level and the number of link branches. After clustering documents, the clustered subtrees are compared at the leaf-node levels using the percentage of matched leaf nodes in the subtrees. The overall complexity of this approach is expressed as $O(N^2)$, where N is the maximum number of nodes in the XML documents. The authors found that when LAX is applied after fragmenting documents, the matched subtrees selected from the output pair

of fragmented documents with a high degree of similarity among trees might not be the subtrees appropriate for integration. To solve this problem, they introduce SLAX (an improved LAX to Integrate XML data at subtree classes) [42]. SLAX divides XML documents into smaller portions by parsing them into K document trees. In each document tree, SLAX applies the weighting factor from LAX to find points for subtree clustering. Since the clustering method relies on the number of link branches and the document's depth, this method may not perform well for deep and complex XML structures.

2.5.2.2 Document List Similarity. Kade and Heuser [33] present an approach called XSIM that uses information from both the structure and the content of XML documents. Three pieces of information permit calculation of the similarity between two nodes of XML trees: the content of the element and the names and path of the nodes. The comparison has two main steps: (1) node matching and (2) document matching. First, the document tree is traversed to produce a set of tuples containing path and content and called a document list for subtrees. Second, the tuples of the document lists are compared and searched for matching nodes based on similarity of textual content, node label and node path. The similarity between two elements is computed as the average of textual content, element name, and path similarity values without considering semantics.

2.5.2.3 Probabilistic Model. In the work of de Keijzer [22], the uncertainty is stored in order to support unattended information integration in probabilistic form using a probabilistic database approach. A problem in using probabilistic databases for data integration is how to determine the probabilities. Many schema-matching techniques suitable for data integration, however, quantify the degree of matching. For example, instance-based matchers use classification techniques. If two data items from different information sources referring to the same real-world object conflict on some attribute value, and one of those values is classified with less certainty that the other in the class corresponding to the attribute, then that attribute value is less likely to be correct and should receive a smaller probability. The same holds for techniques that use dictionaries or thesauri: if a possible data value is not present in the corresponding dictionary, it should receive a smaller probability. The document tree contains two new kinds of nodes: (1) probability nodes and (2) possibility nodes. Comparison is based on the probability

associated with possibility nodes that can compute a possible representation of the matched real-world object. To determine the probability that two XML elements refer to the same real-world object, knowledge rules are applied. These rules can be generic or domain-specific. The amount of uncertainty can be reduced after applying the rules.

Another approach that follows a probabilistic model uses a Bayesian network. Bayesian networks can be applied effectively to detect duplicates in hierarchical and semi-structured XML data. This approach combines the probabilities that children and descendents in a given pair of XML subtrees are duplicates. To compare two candidate XML elements, a maximum overlay between the two trees is computed. Two nonleaf nodes can be matched if they are ancestors of two matched leaves. Once a maximum overlay has been determined, its cost is computed by a string distance function. Leitão, Calado, and Weis [37] present results showing that the model provides great flexibility in its configuration, allowing the use of various similarity measures for the field value and various conditional probabilities to combine the similarity probabilities of the XML elements. The primary disadvantage of Bayesian techniques is their computational complexity.

2.5.2.4 Object Description Similarity. Weis and Naumann [69] propose a method called DogmatiX for comparing XML elements based on their data values, parents, children, and structure. The method comprises three steps: (1) candidate detection that specifies what objects to compare, (2) object identification defining what information is part of a candidate's description, and (3) similarity measure. The method starts by taking an XML document, its XML schema, and a file describing a mapping of element XPaths to a real world for candidate detection. Objects, or elements, are then described. An object description comprises a set of tuples (name, value) that can be identified by heuristics and conditions. Heuristics include r-distant ancestors, r-distant descendants, and k-closest descendants. The conditions that can be used to refine the selection descriptions are content model, string data type, mandatory elements, and singleton elements. The last step is similarity comparison. Similarities in textual values are compared using a variation of string edit distance. Element similarities are evaluated by a variation of the inverse document frequency (IDF) score. Experimental results show that DogmatiX is effective in identifying real and synthetic duplicate XML elements and

documents. This method relies on manual mapping between the elements of schema and real-world entities.

2.5.2.5 Tree Serialization Similarity. Wen, Amagasa, and Kitagawa [70] have proposed an approach to tree serialization similarity developed in the context of data integration. Because the tree structure representation of XML data makes it difficult to measure similarity, it is converted into node sequences by traversing the tree in a particular order (e.g., pre- and post-order). After serialization, the XML data becomes one long node sequence: The sequence is extracted into sub-sequence corresponding to the XML subtree using parameters such as the smallest number of the text nodes, the maximum number of text nodes, and the least height from the leaf node that a subsequence should have. The similarity measure takes into account comparison of textual information using Jaccard similarity and comparison of structural information by edit similarity. The comparison process is accelerated by a Bloom filter [29] providing a probabilistic way to determine if an element is a member of a given set. The authors state that the results are accurate and effective; however, they do not compare their approach with other existing approaches.

2.5.3. Similarity of Collection of Values. Dorneles et al. [24] propose a set of similarity metrics for manipulating collections of values occurring in XML documents. XML nodes can be considered atomic, containing single values such as numbers, texts, and dates, or complex, containing nested node structures. In addition, the authors devide the complex nodes into two categories: (1) tuple elements and (2) collection elements. A tuple element contains multiple sub-elements, but a collection element contains a duplicate of the sub-element. The similarities measure atomic and complex values recursively. The evaluation of XML element similarities requires that the elements to be compared share the same contexts and have similar children.

2.6. XML STORAGE

Storage of XML data in a relational model to identify similarities presents issues of scalability related to the main memory, which affects all other schemes. XRel (a path-

based approach to storage and retrieval of XML documents using relational dabases) [73] uses a single relational schema to store XML documents irrespective of DTDs or XML schema. The topology of XML trees and nodes is represented by the combination of simple path expressions and regions.

The basic XRel schema uses paths as a unit of decomposition of XML trees. It uses four relations, element, attribute, text and path (shown in Table 2.1), to store the data and structure of the XML documents.

Table 2.1. XRel relations

Database relations	
Path	(pathid, pathexp)
Element	(docid, pathid, start, end, index, reindex)
Text	(docid, pathid, start, end, value)
Attribute	(docid, pathid, start, end, value)
Field descriptions	
docid	Document ID
pathid	Path expression ID
pathexp	Path expressions of XML elements
start	Start value of the region
end	End value of the region
index	Forward index
reindex	Reverse index
value	Leaf node and attribute values

The XML document shown in Figure 2.1 provides an example. It can be stored in the relations as shown in Table 2.2. The node ‘author’ is a descendant of the ‘authors’ node, which is a descendant of the ‘book’ node. XRel will store the ‘author’ node signature as ‘#/book#/authors#/author’. Since several nodes may share the same path, storing the simple path expression may lead to a loss of the precedence relationship among nodes. Therefore, the regions of all the nodes help preserve the topology of XML documents.

Table 2.2: XRel tables

(a) Path Table

PATHEXP	PATHID
#/book	1
#/book#/title	2
#/book#/authors	3
#/book#/authors#/author	4

(b) Element Table

DOCID	PATHID	IDX	REIDX	ST	ED
1	1	1	1	5	137
1	2	1	1	15	39
1	3	1	1	44	128
1	4	1	2	60	83
1	4	2	1	91	114

(c) Text Table

DOCID	PATHID	VALUE	ST	ED
1	2	XML in Use	22	31
1	4	Author1	68	74
1	4	Author2	99	105

2.7. XML KEYS

Using keys is another technique to improve matching efficiency. Keys are an essential part of database design; they are fundamental to data models and conceptual design. If keys can be identified for XML documents, the matching process takes dramatically less time [13]. Since most XML documents are data-centric and derived from a relational data model, keys can better identify similarity among subtrees.

PAPER

I. XML DATA INTEGRATION BASED ON CONTENT AND STRUCTURE SIMILARITY USING KEYS

Waraporn Viyanon[1], Sanjay K. Madria[1] and Sourav S. Bhowmick[2]

[1]Department of Computer Science, Missouri University of Science and Technology
Rolla, Missouri, USA

[2]School of Computer Engineering, Nanyang Technological University, Singapore
{wvz7b@mst.edu, madrias@mst.edu, assourav@ntu.edu.sg}

Abstract. This paper proposes a technique for approximately matching XML data based on content and structure by detecting the similarity of subtrees clustered semantically using *leaf-node parents*. The leaf-node parents are considered a root of a subtree, which is then recursively traversed bottom-up for matching. First, the key is used to match subtrees, thus reducing dramatically the number of comparisons. Second, the degree of similarity is measured based on the data and structures of the two XML documents. The results show that this approach finds much more accurate matches with or without the keys in the XML subtrees. Other approaches experience problems with similarity matching thresholds because they either ignore semantic information available or have problems handling complex XML data.

Keywords: XML, Similarity, keys, clustering

1. INTRODUCTION

Data such as ACM SIGMOD Record [9] and DBLP [10] are published and shared using XML. Although the content of these sources is similar, it is described using

different tag names and structures. Researchers have proposed several methods [1, 4, 5, 6] to measure the similarity of XML content or structure. These methods extract several features or keywords of each document and store them in an XML tree. The similarity between XML documents is then calculated by computing the edit distances between two trees, a time-consuming task [1]. The method proposed by Liang and Yokota [4] is similar; it uses a brute-force algorithm to compare the degree of path similarity. On the other hand, some approaches like LAX [5] and SLAX [6], use the characteristics of XML documents, such as depth and number of instances contained, to cluster the documents into subtrees, which are then used to calculate the similarity. Although these methods outperform schemes based on edit distance, they ignore available semantic information such as keys; rather, they rely on finding subtrees or "clustering points," an approach that does not work for all XML data.

Buneman et al. [2] introduced the concept of keys for XML documents. This paper proposes a new approach called XML document integration, or XdoI, that considers both the data structure and the content to match XML documents approximately and thus to integrate XML data sources. More specifically, this approach detects the similarity of two semantically clustered subtrees from two XML documents, taking advantage of XML keys to match the subtrees from the bottom up. This method dramatically reduces the number of comparisons. Once the subtrees are matched based on keys, the remaining unmatched subtrees are processed to find similarities in both their content and structure and thus to select the best matched subtrees. XdoI outperforms LAX and SLAX schemes in terms of false positives, quality of results, and execution time.

2. RELATED WORK

Buttler [3] summarized three approaches to evaluating structural similarity: (1) tag similarity, (2) tree edit distance (TED), and (3) Fourier transform similarity. The tag similarity algorithm is not sufficiently accurate because pages conforming to the same schema, such as HTML, have only a limited number of tags; one page may contain many incidences of a particular tag, whereas the comparison page may contain relatively few

occurrences of the tag. Fourier transform similarity [7] seeks to remove all information from a document except its start and end tags, leaving a skeleton that represents the structure. Buttler [3] proved that this is the least accurate of any of approximation algorithm, and the slowest. In the same work, he also showed that path similarity measures are expensive to compute because there are n-factorial mappings between the paths of two trees.

3. MOTIVATION AND PROBLEM STATEMENT

The LAX approach [5] divides XML trees into subtrees by identifying a clustering point from the height (i.e., the distance from the furthest child) and the number of link branches on such trees. A link branch is a link between two candidate elements that have at least two children or the distance of which to its furthest child is at least three. The subtree can be generated by deleting the link branch below the clustering point. The clustering point is calculated from the maximum weighting factor of the multiplication between the height level and the number of link branches. For example, tinyDB (2,3), shown in Figure 1, has a weight of 6.

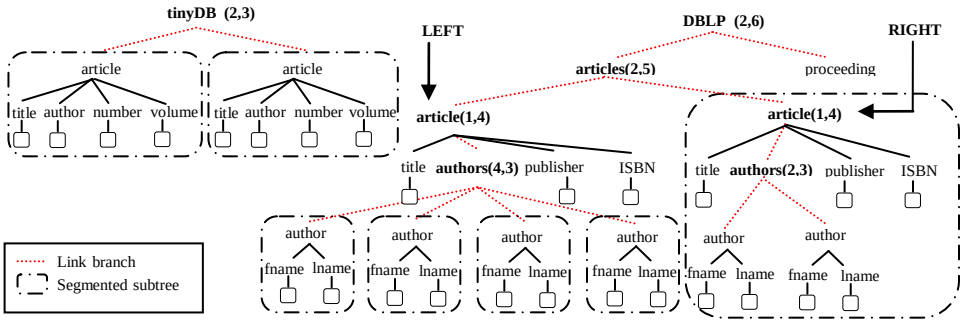


Figure 1. Example of LAX clustering on two different XML structures: DBLP tree has two article elements, part A on the left and part B on the right.

Figure 1 shows LAX clustering on two different XML documents with dissimilar structures. The clustering points on the tinyDB tree are the link branches under the candidate element tinyDB(2,3) because the maximum weight is equal to 2*3, or 6. Therefore, the tinyDB tree is clustered into two subtrees rooted by the article nodes, but the DBLP tree is clustered into four subtrees rooted by the author nodes for the first article since the maximum weight is from the element authors(1,3). Figure 1 shows clearly that the resulting subtrees in both documents are semantically different. The first tree is cut into a set of articles, but the second (part A) is chopped into a set of authors. The results of clustering affect subsequent steps such as subtree matching because they permit comparison of various types of objects. In addition, the leafnodes "title", "publisher," and "ISBN" of the first article on the second tree are not considered in any subtree.

The second article on the second XML tree in Figure 1 (part B) is clustered into one subtree because the maximum weight is 2*5, or 10, as calculated from the element article(2,5). This example shows that an XML document may yield various kinds of subtrees and therefore, similarity scores will vary. Elements missing from the first article, such as "title," "publisher," and "ISBN," also occur after clustering.

Liang and Yokota [6] found that when LAX is applied after fragmenting documents, the matched subtrees selected from the output pair of fragment documents with a high degree of tree similarity might not be the appropriate subtrees for integration. SLAX [6] is an improved LAX that solves this problem. It divides XML documents into smaller portions by parsing them into K document trees. In each document tree, SLAX applies the weighting factor from LAX to find points for subtree clustering. Thus, they solve the issue of matching right subtrees but the problem with LAX clustering discussed above still occurs in divided trees. We elaborate this problem further in Section 5.

This paper addresses the drawbacks of clustering discussed above using leaf-node parents as clustering points, exploiting keys, and employing a bottom-up approach to match subtrees recursively from the bottom-up.

4. APPROACH

The XDoI approach involves three phases. In Phase I, the base XML tree and target XML tree are clustered by treating leaf-node parents as clustering points. The clustered subtrees in the base XML tree are considered independent items to be matched in Phase II with clustered subtrees in the target XML tree. In Phase III, the best matched subtrees are integrated in the first XML tree.

4.1 Basic Definitions

Definition 1. XML Document Tree: An XML document tree T is an ordered labeled tree parsed from an XML document. The terms T_b and T_t represent two XML document trees, where b denotes the base tree and t denotes the target tree. The T_b and T_t are clustered into subtrees.

Definition 2. Leaf-node parent: A leaf-node parent is a node that has at least one child leaf node. It is considered the root of a subtree in the clustering process.

Definition 3. Clustering point: The clustering point is the link above the leaf-node parent. It indicates the place for clustering an XML tree into subtree(s).

Definition 4. Simple subtree: A simple subtree is a clustered tree with only a root and leaf nodes.

Definition 5. Complex subtree: A complex subtree is a clustered subtree with at least one simple subtree, a root, and one or more of leaf nodes.

4.2 Preprocessing

The XML documents are compared by using XRel [8] to parse and store them in relational tables based on their structure. Storing XML data in a relational model to evaluate similarity addresses the issue of scalability, which affects all other schemes. XRel uses four relations (element, attribute, text, and path) to store the data and structure of the XML documents and a document relation to store the complete XML document, as shown in Table 1.

This paper describes the process for identifying a leaf-node value match for all unique node values that share a single path using the SQL statement shown in Table 2. The unique leaf node is considered the key that can identify the subtree. Some XML documents may not contain a key in some subtrees, or an item in either the base or target

XML tree may have a key but may not appear in the other XML tree. Therefore, the best match for this case must be identified by comparing the remaining subtrees.

Table 1. XRel relational schema and subtree table

Element(docid, pathid, start, end, index, re-index) **Attribute**(docid, pathid, start, end, value) **Text**(docid, pathid, start, end, value) **Path**(pathid, pathexp) **Document**(docid, document) **Subtree**(docid, subtreeid, pathid, start, end, key, value)

Table 2. Finding keys

```
SELECT docid, pathid, value
FROM subtree
GROUP BY docid, PathID,Value
HAVING Count(Value) = 1
```

4.3 Phase I: Clustering an XML Tree into Subtrees

An XML tree can be parsed into small independent items by clustering it into more meaningful subtrees. Each clustered subtree represents independent items. As explained by Lian and Yokota [5], a well clustered subtree requires that (1) each subtree represents one independent item, (2) each independent item is clustered into one subtree, and (3) the leaf nodes belonging to that item are included in the subtree.

In the approach presented here, an XML tree is clustered into an independent item using leaf-node parents as clustering points. The leaf-node parents are considered a root of each subtree. The clustered subtrees are categorized as either simple or complex. They are stored in the subtree table (see Table 1) to be used later in subtree matching.

4.4 Phase II: Matching Subtrees

The keys in the base subtrees are matched with the keys in the target subtrees. This matching process reduces the number of unnecessary subtree matches. One-to-multiple matches may occur in this step. Once the degree of similarity is determined, the number of matches is reduced. The unmatched subtrees are also needed to determine the degree of similarity in each subtree pair.

To find the correct matched subtree, both the content and structure of the base and target XML trees are considered by comparing the PCDATA (parsed character data) values and signatures. First, the subtree similarity degree (SSD) is determined as follows:

Definition 6.1. Subtree Similarity Degree (measure1): Let t_{bi} be the base subtree and t_{tj} be the target subtree. Assume n is the number of leaf nodes having the same PCDATA value. Let n_{bi} represent the number of leaf nodes in t_{bi}, and let n_{tj} represent the number of leaf nodes in t_{tj}. For scoring purposes, each common node is assigned 1 point, and a common node defined as a key is assigned 2 points:

$$s_1(t_{bi}, t_{tj}) = \frac{n}{n_{bi}} \times 100\%. \quad (1)$$

Definition 6.2. Subtree Similarity Degree (measure2): This metric is the ratio of common matched leaf-node values in the base subtree to the same values in the target subtree:

$$s_2(t_{bi}, t_{tj}) = \frac{2 \times n}{n_{bi} + n_{tj}} \times 100\%. \quad (2)$$

This measure eliminates the number of one-to-multiple matches having the maximum SSD (measure1) as overlapping target subtrees.

Definition 7. Matched Subtree: The matched pair of subtrees t_{bi} and t_{tj} is the pair that has the maximum subtree similarity degree based on Definitions 6.1 and 6.2. The maximum subtree similarity degree is considered as a matched subtree:

$$SM[i] = Max\left(s_1(t_{bi}, t_{tj})\right). \quad (3)$$

Definition 8. Path Similarity Degree (PSD): PSD is the best matched subtree identified by comparing the number of nodes in the base path to the number of nodes in the target path on the matched leaves that have the same labels, excluding leaf nodes. This metric is applied when the subtrees have the same maximum degree of similarity as mentioned in Definition 7:

$$PSD(i) = \frac{N}{N_{bi}} \times 100\% \quad (4)$$

where *N* denotes the number of nodes in the base path that have the same labels as those in the target path, N_{bi} denotes the total number of nodes in the base path, and i is the total number of matched leaf node paths in the base subtree between 1 to *k* paths.

Definition 9. Path Subtree Similarity Degree (PSSD): After PSD is calculated as in Definition 8, the PSSD is calculated as

$$PSSD(t_{bi}, t_{tj}) = \frac{\sum_{i=1}^{k} PSD(i)}{k} \times s_1(t_{bi}, t_{tj}) \times 100\%. \quad (5)$$

4.5 Phase III: Join Algorithm

Here S_b and S_t are two XML data sources, and the XML document $d_b \in S_b$ and $d_t \in S_t$ be clustered into XML document trees T_b and T_t. These trees, in turn, are clustered into subtrees, t_{bi} and t_{tj}, where i denotes the number of subtrees in the base document tree and j denotes the number of subtrees in the target document tree. The steps to join two XML documents are as follows: (1) Find the degree of similarity for each subtree pair. If there is more than one matched subtree, find the maximum degree of similarity among them. (2) Calculate the degree of similarity between two trees from the mean value of the degree of similarity degree between the matched subtrees. (3) If the degree of similarity between two trees is greater than a given threshold τ, where $0 \leq \tau \leq 1$, the two documents can be integrated at the clustering point.

4.6 Algorithm

The approach presented here can be written in pseudocode as follows: The algorithm is processed after the XML documents are parsed into a relational database. There are four main modules: (1) preprocessing, (2) clustering, (3) matching subtrees, and (4) integrating matched subtrees. In the subtree matching phase, the subtrees are first matched with keys according to Query 3 in Table 3. The first unmatched subtree in the first XML document is compared with all the subtrees in the second XML document. This procedure is performed recursively for all the subtrees in the first XML document. The best match can be calculated by following the algorithm steps shown in Figure 2.

```
Algorithm XDoI: Input: XML documents d_b and d_t and Output: Set of matched subtrees pairs (t_bi, t_tj)
//Module1 Identifying key(s)
Define key();                                   //Table 2: Finding key(s)
//Module 2: Clustering the XML trees
Find_leafnode_parent();                         //Sub Module
ClusterXMLTree();                               //Sub Module
//Module 3: Matching subtrees
Match_with_key();                               //Table 3: (Query 3)
// Subtree Similarity degree computation
for (every t_bi in d_b) {                       //Subtrees from the based document
  MaxSim[i] = 0;
  for (t_tj in d_t){
    CalSimilarity S(t_bi ,t_tj)                 //Definition 6.1&2
    MaxSim[i] = Max(MaxSim[i], S(t_bi ,t_tj));  //Definition 7
  }
  StoreMSSD(t_bi, t_tj, MaxSim[i]);             //Store MSSD in a temp table
}                                               // Path Subtree Similarity degree computation
for (every t_bi in MSSD, such that Count MaxSim() >1 and MaxSim > τ ) {
  MaxPath[i] = 0                                //Match subtree more than one pair
  for (j = 1 to k_t) {
    CalPathSimilarity P(t_bi ,t_tj)             //Definition 8
    MaxPath[i] = Max(MaxPath[i], P(t_bi ,t_tj)); //Definition 9
  }
  StorePSSD(t_bi, t_tj, MaxPath[i]);            //Store MSSD in a temp table
}
//Module4: Integration
// similarity degree > the threshold
for (every t_bi in PSSD, such that MaxPath > τ ){
  d_i = integrate(S_b, S_t)                     // Section 4.5
  return (d_i);
}
Sub Modules:
Find_leafnode_parent(){
 for every p_i from the PATH table{
  if lastpathsection(p_i) is not attribute {
    lnp = Remove the last path section from (p_i);
    store_lnp(lnp);                             //store a leafnode parent into a temporary table
  }
 }
}
ClusterXMLTree()  {
 for (i in all_lnp){
   region_i = Retrieve leafnode parent info     //Table 3 (Query 1)
   t_i = find_subtree(region_i);                //Table 3 (Query 2)
   store_subtree(t_i)
 }
}
```

Figure 2. XDoI Algorithm

Table 3. SQLs for clustering subtrees and matching subtrees with key

Query1: Retrieve leafnode parent information: **Select** distinct e.docid, e.pathid, e.st, e.ed **From** tmp_leafnode_parent p , element e **Where** p.docid = e.docid and p.ppathid = e.pathid
Query2: Find subtrees: **Select** docid, pathid, st, ed, ++subtreeid, value **From** txt **Where** st >= region.st and ed <= region.ed
Query 3: Match subtrees with keys: **Select** s1.subtreeid, s2.subtreeid **From** subtree s1, subtree s2 **Where** s1.docid = 1 and s2.docid = 2 and s1.key = 'Y' and s2.key = 'Y' and s1.value = s2.value

5. PERFORMANCE EVALUATION

This section describes experiments conducted to evaluate the efficiency and effectiveness of this algorithm compared with LAX and SLAX [5, 6]. The experiments used on Intel Pentium 4 CPU 2.80GHz with 1GB of RAM running on Window XP Professional with Sun JDK 1.6.0_02 and Oracle Database 10g Standard Edition. SIGMOD Record [9], 482 KB, was used as the base document, and three segmented documents of DBLP.xml [10], 700 KB each were used as the target documents. Some synthetic XML datasets classified according to key, structure type (shallow or deep), and file size were also used.

5.1 Experimental Results

First, the variation of clustered subtrees among all three algorithms was evaluated. The clustering points (subtree roots) and the number of subtrees were then compared using SIGMOD Record and DBLP. Table 4 shows the difference IN the clustered subtree AMONG the three approaches. XDoI clustered subtrees by leaf-node parents, which covering all leaf-node values on the documents. This guaranteed that the associated values were not missed while clustering. LAX and SLAX were clustered according to the weighting factor *w* discussed in Section 3. The clustered subtrees from SLAX rely on the *K* value mentioned in that section. Clustering of SIGMOD Record yielded segmented

subtrees at three different levels, “issue,” “article,” and “author,” that can be integrated with various kinds of bibliographical documents. On the other hand, LAX yielded a subtrees only at “issue” level, which affected the similarity scores when the subtrees clustered using LAX were compared with the subtrees rooted at “article” level from the DBLP dataset. SLAX clusters XML documents based on the selected *K* value; however, the appropriate *K* value required for clustering across multiple XML documents was not clear [6] because no procedure was provided to select this value. This uncertainty in *K* value also causes the loss of some meaningful information such as “issue volume” and “issue number” when *K* is greater than 4 for SIGMOD Record. Similarly, *K* has different values for different documents, as shown in Table 4. Therefore, clustering using LAX and SLAX can substantially affect the degree of subtree similarity identified and thus result in the integration of mismatched subtrees.

Table 4. Clustered point and number of subtrees yielded by each approach

XML document	**XDoI**		**LAX**		**SLAX**		
	subtree-root element	**# of subtrees**	**subtree-root element**	**# of subtrees**	***K***	**subtree-root element**	**# of subtrees**
SIGMOD Record	issue article authors	67 1504 1504	issue	67	k≤4 k>4	issue article	67 1504
DBLP (dblp01.xml)	inproceed-ings	769	inproceed-ings	769	Any k	inproceed-ings	769

The next steps were to evaluate the execution time for clustering subtrees and to identify keys. The degree of subtree and path similarity was then computed for three pairs of SIGMOD Record and selected DBLP fragments respectively, labeled 1st, 2nd, and 3rd. The experiments used a threshold value of 0.5. The three data set pairs had no predefined keys, so the key identification module was required. However, integration of two XML documents, such as those generated by RDBMS, with predefined keys would not require this module. In the case of hybrid XML documents, some portions of which have predefined keys, execution time for the key identification module would be reduced. The

approach tested here requires both key identification and key mapping modules; nonetheless, the execution time was less than that for SLAX because SLAX clusters the XML documents into subtrees using the weighting factor. Most of the execution time, therefore, is used to calculate this factor. The key mapping module in XDoI, on the other hand, reduces the number of subtrees needed to calculate SSD and PSSD.

These experiments also compared this new approach with and without keys to observe the improvement in overall execution time produced by the use of keys. In Table 5, the numbers in parentheses represent XDoI execution time without keys. Obviously, this time is longer because SSD must be calculated for each subtree pair. Figure 3 compares XDoI and SLAX for each module.

The effectiveness of each approach was evaluated by measuring the false positives based on the number of matched pairs and the number of actual matched subtrees. Table 6 shows the number of matched subtrees yielded by these experiments; the values in parentheses indicate the number of correctly matched subtrees and the number of incorrectly matched subtrees, respectively. The results indicate that all the incorrect matched pairs are simple subtrees rooted by the “authors” element and overlapped with a complex subtree rooted by the “article” element. Each pair is matched with a target subtree because the base and target subtrees share the same authors but they are from different articles. The false positive rates show that the new approach, with keys, has a much lower false positive rate than SLAX. Therefore, it can detect the matched subtrees appropriate for integrating XML documents more precisely than SLAX.

5.2 Results Quality

This section compares the accuracy of the degree of similarity generated by the new approach and by SLAX for several types of XML files. The quality of the subtree matching result is characterized as *Sn/An*, where *Sn* is the number of matched subtrees yielded by a given approach and *An* is the number of actual matched subtrees. Figure 4 shows the quality of results for XML documents with file sizes ranging from 1KB to 71KB. It demonstrates that the new approach and SLAX perform similarly for shallow and semi-shallow XML documents. For large and deep XML documents, however, XDoI yields much better results than SLAX. The performance of SLAX drops from 100% to

0% because it does not cluster the complex XML documents into appropriate subtrees, for the reasons discussed in Section 3.

Table 5. Execution time (in seconds) for clustering and key generation in SIGMOD Record and DBLP data

Module	**XDoI**			**SLAX**		
	1st pair	**2nd pair**	**3rd pair**	**1st pair**	**2nd pair**	**3rd pair**
Clustering	17625	20070	22344	233482	281576	258624
Finding keys	156811	154796	184983	-	-	-
Mapping with keys	217358	209748	89484	-	-	-
SSD	15975258 (37149128)	17697580 (45629554)	53201074 (59464448)	17775897	20445114	36746179
PSSD	162200 (185655)	32063 (40594)	890605 (1082337)	962885	935901	1385309
Total Time	16529252 (37352408)	18114257 (45690218)	54388490 (60569129)	18972264	21662591	38390112

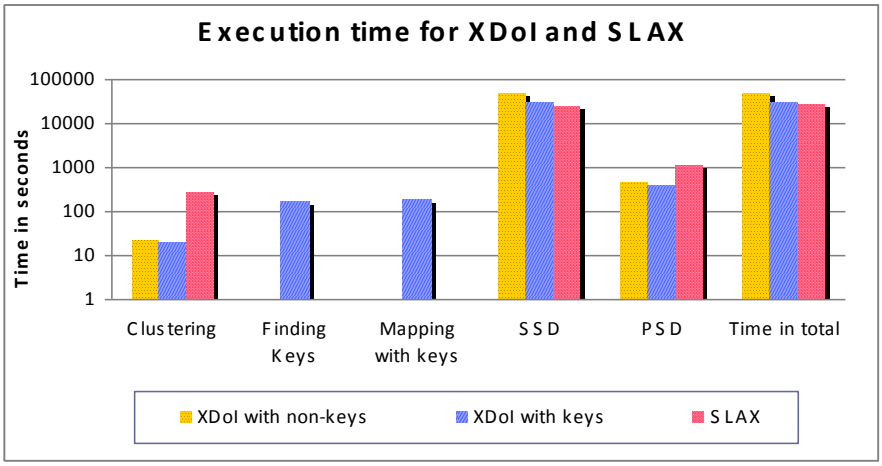


Figure 3. Average execution time for XDoI and SLAX

Table 6. Matched subtrees of SIGMOD Record and DBLP

Threshold	XDoI with keys			SLAX		
	1st pair	2nd pair	3rd pair	1st pair	2nd pair	3rd pair
50%	361(339,22) FP = 6.09%	336(322,14) FP = 4.17%	93(67,26) FP=27.96%	314(273,41) FP=13.06%	286(225,61) FP= 21.33%	102(55,47) FP=46.08%

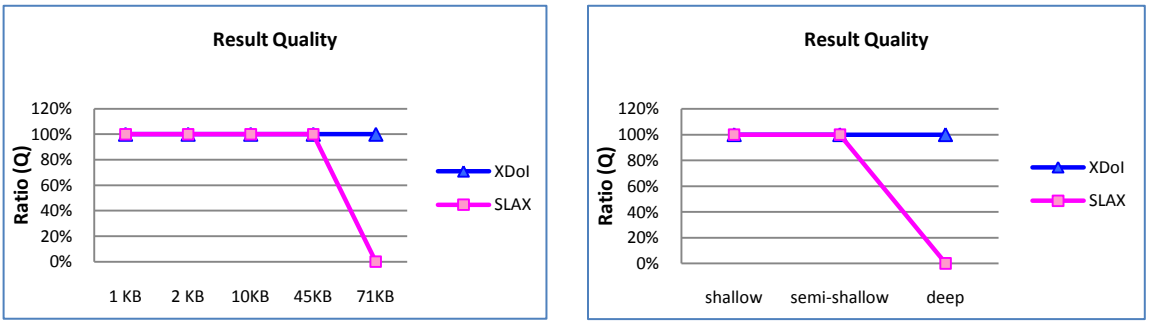


Figure 4. Quality of results various (a) file sizes and (b) file types

6. CONCLUSIONS

This paper presented a data-centric approach to clustering XML documents into subtrees using leaf-node parents and keys to reduce the number of subtrees matches and improve the determination of similarity by reducing false positives. The performance evaluation indicates that keys are very efficient at identifying appropriate subtrees matches among XML documents. The XDoI approach performs better than SLAX and LAX for complex XML documents because they could not identify appropriate subtrees in the step of subtree clustering.

7. REFERENCES

[1] Bille, P.: Tree Edit Distance, Alignment Distance and Inclusion. ISBN 87-7949-032-8

[2] Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. Computer Networks, Volume 39, Issue 5, August 2002, pp 473 - 487.

[3] Buttler, D.: A Short Survey of Document Structure Similarity Algorithms. In: International Conference on Internet Computing 2004, pages 3-9, 2004.

[4] Liang, W., Yokota, H.: A Path-sequence Based Discrimination for Subtree Matching in Approximate XML Joins. In: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06), IEEE, pp. 23-28, 2006.

[5] Liang, W., Yokota, H.: LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration. In: Proceedings of BNCOD 2005, LNCS3567, Springer, pages 82-97, July 2005.

[6] Liang, W., Yokota, H.: SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes. In: Proceedings of DBWeb2005, IPSJ Symposium Series. 2005(16), pp.41-48.

[7] Rafiei, D.: Fourier-Transform Based Techniques in Efficient Retrieval of Similar Time Sequences. Thesis of University of Toronto, 1999.

[8] Yoshikawa, M., Amagasa, T.: XRel: A Path-based Approach to Storage and Retrieval of XML Documents. In: Proceedings of the 19th IEEE International Conference of Data Engineering (ICDE), India, pp. 519-530, 2003.

[9] ACM SIGMOD Record in XML, http://www.acm.org/sigmod/record/xml XML Version of DBLP, http://dblp.uni-trier.de/xml/

II. A SYSTEM FOR DETECTING XML SIMILARITY IN CONTENT AND STRUCTURE USING A RELATIONAL DATABASE

Waraporn Viyanon
Department of Computer Science,
Missouri University of Science and Technology
Rolla, Missouri, USA
wvz7b@mst.edu

Sanjay Kumar Madria
Department of Computer Science,
Missouri University of Science and Technology
Rolla, Missouri, USA
madrias@mst.edu

ABSTRACT

This paper describes a technique that uses keys to detect similarities in structure and content between two XML documents. The technique has three major components: a subtree generator and validator, a key generator, and similarity detection components that compare the content and structure of documents. First, an XML document is stored in a relational database and extracted into small subtrees using leaf-node parents. These parents are considered the root of a subtree, which is then recursively traversed from the bottom up. Second, potential keys are identified to facilitate efficient matching of XML subtrees from the two documents. Key matching dramatically reduces the number of comparisons dramatically. In addition, the number of subtrees to be processed is reduced in the subtree validation phase using instance statistics and a taxonomic analyzer. First, the subtrees are matched by the keys, then the remaining subtrees are matched by determining degrees of similarity in content and structure. To improve the results of comparisons, XML element names are transformed according to their semantic similarity. The results show that this method selects the clustering points appropriately and dramatically reduces the overall execution time.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—Data Mining

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—Clustering, Information Filtering, Selection process

General Terms

Measurement, Performance, Verification.

Keywords

XML, Similarity measures, Keys, Clustering, Taxonomy Analyzer

1. INTRODUCTION

XML is a standard for data representation and interchanging data on the Internet because it can represent data from a wide variety of sources. Data such as DBLP [16] and ACM SIGMOD Record [1] are published and shared on the Internet using the XML format. XML eases the integration of data from multiple sources; however, correlating XML data sources presents significant complexities due to the structure of XML documents. Different data sources may have similar content; they may be described using different tag names and structures [1, 3].

This paper describes improvements developed for XML document integration (XDoI) [13]. The approach presented here considers both the structure and content of data to match XML documents approximately and thus to integrate data sources using keys. XDoI clusters an XML document into smaller subtrees, considered individual objects, using leaf-node parents. These parents can generate many clustered subtrees due to overlap among the subtrees. A large number of clustered subtrees extends computational time needed to compare subtrees, identify similarities among them, and match them appropriately. To eliminate unnecessary subtrees and thus reduce the number of comparisons necessary, this approach uses a taxonomic analyzer to determine how close the meanings of element names are and to transform them into a single category. For example, an XML document containing an element name “Pages” may have two

subelements, “initPage” and “endPage,” as descendants. These three element names can be categorized in a single group using the taxonomic analyzer. The concept of instance statistics [14] is used to determine the relationships among element names to eliminate subtrees that do not hold a one-to-one relationship among XML elements. Keys are used to match subtrees and the results of the matching are analyzed to identify inappropriate subtrees that are not irrelevant to the subtrees in the other document, thus reducing the number of comparisons necessary to identify similarities. Experiments demonstrate that the system improves overall computation without sacrificing accuracy.

The remainder of the paper is organized as follows: Section 2 presents an overview of XML document integration. Section 3 describes previous work of the subject. Section 4 introduces the improved approach and describes the system architecture in detail. The algorithm for this approach is defined in Section 5. Section 6 presents the performance results, and Section 7 offers conclusions.

2. BACKGROUND

XML documents can be considered collections of objects. A scalable integration technique is needed to accommodate the growing number of XML data sources. XML integration generally begins by extracting XML documents into subtrees according to their semantics. The subtrees are represented as individual objects. The clustered subtrees are evaluated in order to detect the similarity among them. Similar subtrees in structure and semantics are integrated.

Clustering XML documents automatically into proper objects is challenging. Existing techniques for clustering include LAX [9], SLAX [10], and S-GRACE [7]. After XML documents are clustered into subtrees, they are compared to identify similarities. The result may consider similarities of: (1) structure, (2) content, or (3) both structure and content. Previous research [2, 3] has addressed structural similarity, aiming to extract from documents pure structural information. Tree edit distance (TED) measures the minimum number of node insertions, deletions, and substitutions required to convert one tree into another. By default, TED assigns a unit cost to each edit operation. The edit

distance between two trees is the smallest cost of transforming tree T_1 to tree T_2 [18]. This transformation process is computationally very expensive and leads to a prohibitively high run time. For a tree with *n* nodes, *l* leaf, and a depth of *d*, the computational time is expressed as $O(n^2 min^2(l, d))$ time and $O(n^2)$ space. Thus, it is not practical for similarity matching over large XML data repositories.

Traditional content similarity methods can be roughly separated into two groups: character-based techniques and vector-space-based techniques. TED can be applied to measure context similarity using a character-based technique. This technique relies on character edit operations. It transforms strings into vector representation on which similarity is computed.

Content similarity can be measured based on the information content of the least common subsumer (LCS) of concepts A and B in a hierarchy. LCS is the most specific concept that is an ancestor of both concepts A and B. The measures, based on the information content, are Resnik [12], Jiang [6] and Lin [11]. All three required a source for information content for concepts such as WordNet. WordNet [5] is a utility program that allows a user to compute information content values from the Brown Corpus, the Penn Treebank, the British National Corpus, or any given corpus of raw text.

Liang et al. [8] also proposed measurement of structure and content similarity. Their approach, called a path-sequence-based discrimination, solves the problem of the one-to-multiple matching in leaf-clustering-based approximate XML join algorithms. When calculating similarity scores, only identical text nodes are considered, and the number of identical labels is counted. This method decreases the incidence of one-to-multiple matching.

These solutions are easy to implement; however, they have several limitations. For example, computing TED is time-consuming and therefore impractical for similarity matching over large XML data repositories. The same applies to SLAX [10], which uses a brute-force algorithm to determine the degree of path similarity.

3. XDoI

This section presents an overview of the previous approach, XDoI [13]. It then demonstrates how XDoI system outperforms SLAX. Finally, it addresses the drawbacks of the XDoI approach, explaining how it affects the similarity computation.

3.1 XDoI Overview

Previous work [13] proposed the XDoI approach to evaluate the approximate similarity between XML documents. Because many real XML documents are constructed from repeating elements, those to be integrated are fragmented into subtrees representing independent objects. They can be divided into independent subtrees at the repeating elements using leaf-node parents.

The XML keys introduced by Buneman et al. [4] play an essential role in subtree matching. They are used to identify their own subtree. Keys for XML documents are found by identifying a leaf-node value match for all unique node values with the same path.

The subtree similarities are determined first by key matching, then by the degree of similarity between trees. The keys in the base subtrees are matched with the keys in the target subtrees. This process dramatically reduces the number of unnecessary subtree matchings.

The unmatched subtrees are evaluated for the approximate degree of subtree similarity based on the content and structure of XML documents. The matched subtree pairs are determined from the maximum degree of similarity, which is greater than a user defined threshold.

The approach is first to map XML documents to relational databases, then to use SQL queries to execute the following modules: (1) clustering of XML documents into independent subtrees using leaf-node parents, (2) identification of keys, (3) measurement of the degree of similarity in content and structure, and (4) matching of subtrees. The resulting matched subtree pairs remain in the relational databases, which are easy to integrate and output in XML format.

3.2 Comparison with SLAX

XDoI divides XML documents into subtrees in a data-centric manner using leaf-node parents. Previous experimental results have shown that this approach leaves no information mission after clustering. To compare this approach with other work, SLAX [10] has been implemented in a relational database using XRel [17] to support large XML documents. The results showed that SLAX is not suitable for clustering complex XML documents into proper subtrees because its clustering method ignores semantic information. This limitation arises because SLAX define clustering spots base on a weighting factor computed by multiplying the number of link branches and by the depth of the element. This depth is the distance from the element to its furthest child. A link branch is a link between two candidate elements. Either it must have at least two children, or the distance to its furthest child must be at least three. A subtree can be generated by deleting the link branch below the clustering point. The clustering point is calculated from the maximum weighting factor, determined by multiplying the height of the candidate element and by the number of link branches. SLAX may not cluster subtrees appropriately, in which case they cannot be compared with subtrees from the other XML document. Previous work [13] has shown that SLAX cannot cluster XML documents containing deep and complex structures into proper subtrees; therefore, it matches subtree inaccurately.

3.3 Problems of XDoI

Clustering XML documents using leaf-node parents can also produce many subtrees because of subtree overlap. Figure 1 provides an example from two XML documents, one a SIGMOD Record and the other DBLP. It shows the results of subtree clustering using the leaf-node parents, “issue,” “article,” and “authors.” There are three overlapped subtrees in the SIGMOD Record. The “issue” subtree contains two different levels of subtrees, inside which are the subtrees rooted by “article” and those rooted by “authors.” A large number of clustered subtrees increase the cost of computing the degree of similarity.

Automatically elimimating unnecessary subtrees is not simple; selection of subtrees for elimination requires additional information.

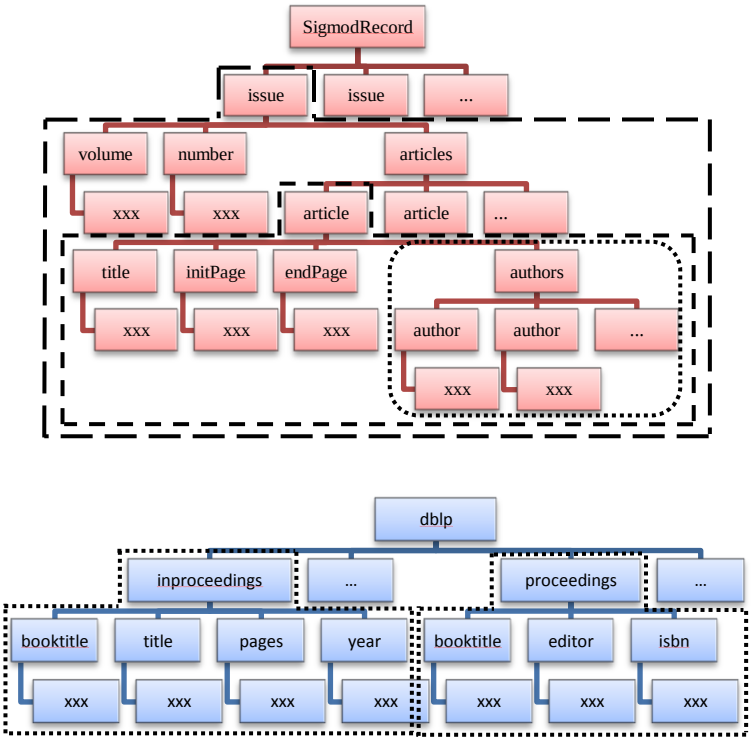


Figure 1. XML documents: (a) SIGMOD Record and (b) DBLP documents

4. XDI-CSSK

This section introduces an improved approach to XML data integration based on content and structure similarity using keys (XDI-CSSK), and it shows how this approach can solve the problem of XDoI discussed in Section 3.3. System design is discussed here as well.

4.1 XDI-CSSK System

The following describes the XDI-CSSK system architecture, which is comprised of four components: (1) XML document storage, (2) subtree generator and validator, (3) XML key generator, and (4) subtree similarity components.

Figure 2 illustrates the system architecture. First, XML documents are stored in a relational database to increase scalability and avoid problems resulting from memory restrictions. Next, XML documents are clustered into subtrees using leaf-node parents. The integrity of the clustered subtrees is verified so that they can be compared in the next

step using the subtree filter. The subtree filter uses the concept of XML instance statistics and a taxonomic analyzer to eliminate inappropriate subtrees before comparing subtrees. After filtering subtrees, the similarity components are used to determine the similarity in terms of content and structure. This process involves three measurements: (1) degree of subtree similarity on based document (SSD1), (2) degree of subtree similarity on both documents (SSD2), and (3) degree of subtree path similarity degree (PSSD). The subtree pairs with the highest degree of similarity are selected as matched pairs for later integration.

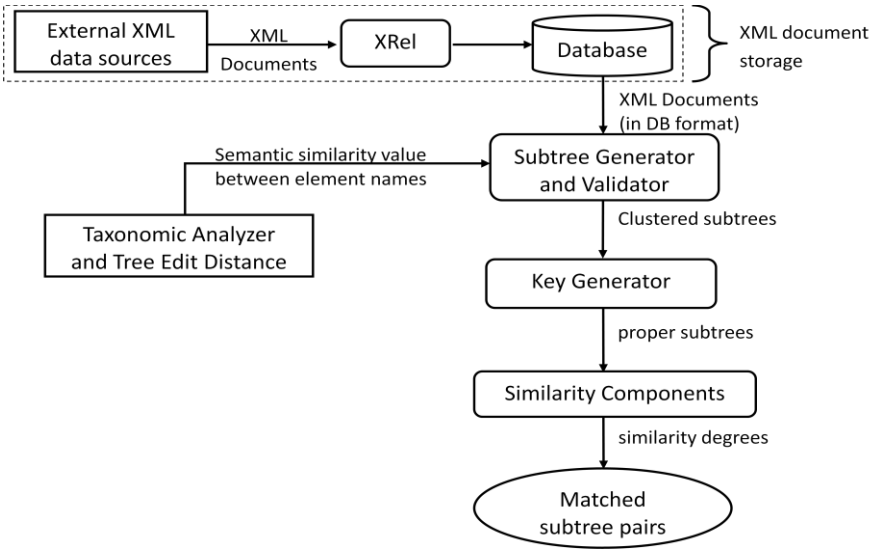


Figure 2. XDI-CSSK system architecture

4.1.1 XML Document Storage

For scalability, the XML documents are loaded into a relational database using XRel [17]. The database is designed to store XML documents and degrees of XML document similarity. XRel decomposes an XML document into nodes on the basis of its tree structure and stores in relational tables according to the node type, with information on path from the root to each node. The basic XRel schema consists of the following four relational schemas shown in Figure 3: *Document*, *Element*, *Attribute*, *Text*, and *Path*:

Document (docID, value)
Element (docID, pathID, start, end, index, reindex)
Attribute (docID, pathID, start, end, value)
Text (docID, pathID, start, end, value)
Path (pathID, pathexp)

Figure 3. XRel relations

The database attributes "docID," "pathID," "start," "end," and "value" represent the document identifier, *simple* path expression identifier, start position of a region, end position of a region, and the string value, respectively. An element node or a leaf node is identified by its region and stored in the relations *Element* and *Text*. To identify each of the attribute nodes, the attribute name is retained as the suffix of the simple path expression of an attribute node, and the attribute value is stored in the relation *Attribute*. The database attribute "pathexp" in the relation *Path* stores simple path expressions.

To store leaf-node parents, clustered subtrees, and degrees of subtree similarity, three additional relations were built; they are shown in Figure 4.

Leanode_parent (docID, ppathExp, ppathid, pathexp, pathid)
Subtree (docID, ppathID, pst, ped,pathid, st, ed, value, key, subtreeid)
Subtree_similarity_score (base_docid, base_subtreeid, target_docid, target_subtreeid, ssd1, ssd2, pssd, match_type)

Figure 4. XDI-CSSK's relations

The *leafnode_parent* relation stores path expressions that have leaf nodes and their parent path expressions. The "pathids" of parent paths can be retrieved from the path relation. The *subtree* relation stores the clustered subtrees, which are used later for comparison. Each subtree contains path information, including the content values at the leaf node level and the key flag. The key flag is used to identify unique leaf nodes. The

subtree_similarity_score relation stores the results of comparison for similarity. The attributes of this measurement are labeled SSD1, SSD2, and PSSD; they are discussed above in Section 4.1. The attribute "match_type" identifies a type of measurement of each subtree pair. This attribute can be either SSD1, SSD2, or PSSD. The similarity measurement functions are discussed in detail in [13].

These three relations reduce the complexity of SQL queries (reducing natural joins, nested sub queries, and correlated sub queries), thus minimizing the size of the SQL queries and improving the performance of the approach.

4.1.2 Subtree Generator and Validator

The subtree generator and validator procedure has three steps: (1) extracting leaf-node parents, (2) validating leaf-node parents using the taxonomy analyzer, and (3) clustering XML documents into subtrees.

The subtree generator is intended to produce small independent items by clustering XML documents into meaningful subtrees. Each clustered subtree represents independent items. A well clustered subtree requires (1) that each subtree represents one independent item, (2) that each independent item be clustered into one subtree, and (3) that the leaf nodes belonging to that item be included in the subtree.

In some circumstances, fragmenting an XML tree into well clustered subtrees is difficult. The nature of XML documents containing content information at the leaf-node level is known; therefore, an XML tree can be more easily clustered into an independent item by using leaf-node parents as clustering points.

4.1.2.1 Leaf-node parent extraction

Because the contents of XML documents are stored at the leaf-node level, leaf-node parents are used as the clustering points to fragment the documents into subtrees and thus to capture the content. The leaf-node parents are found using the SQL query in Figure 5.

The query searches for the paths that contain content at the leaf node. It returns "docid," "pathid," and "pathexp" (i.e., path expressions), which have associated content values. The path expressions returned from the query are trimmed by removing the last

element of each. The trimmed path expressions are called leaf-node parent path expressions. They are stored in the *leafnode_parent* relation.

```
SELECT distinct docid, p.pathid as pathid, pathexp
FROM text l, path p
WHERE p.pathid = l.pathid
```

Figure 5. SQL query for finding leaf-node parents

The leaf-node parents from SIGMOD Record in Figure 1 (a) are as follows:

(1) #/SigmodRecord#/issue,

(2) #/SigmodRecord#/issue#/articles#/article, and

(3) #/SigmodRecord#/issue#/articles#/article#/authors#/author.

The leaf-node parents from DBLP in Figure 1 (b) are:

(1) #/dblp#/inproceedings and

(2) #/dblp#/proceedings.

4.1.2.2 Leaf-node parents filter

Since leaf-node parents function here as clustering points, they become the root of the clustered subtrees. A subtree should contain all the various attributes of an object, not merely one kind of information. The instance statistics concept [14] permits evaluation of the relationship between the leaf-node parent and its children determine whether they preserve a one-to-one relationship. For example, in Figure 1 (a) the “authors” node of a subtree is the parent of two “author” nodes, which are leaf nodes. The “authors” node is considered a root of this subtree. The subtree does not contain a variety of information and it is; therefore, not useful for extracting the extract this kind of subtree that represents an individual object and thus for comparison based on measurements of subtree similarity.

4.1.2.2.1 Relabel XML element names semantically

To determine the variety of information, the relationship between the leaf-node parents and their children is checked. First, the children are evaluated for semantic similarity using Wu and Palmer’s metric [15] to measure the similarity between two

words. This metric takes into account both path length and the depth of the least common subsumer based on the lexical database WordNet [5]. It relies on the following formula:

$$sim(s,t) = \frac{2 * depth(LCS)}{depth(s) + depth(t)}$$

where *s* and *t* denote the source and target words being compared, depth(s) is the shortest distance from the root node to a node *s* on the taxonomy where the synset of *s* lies, and LCS denotes the least common subsumer of the words *s* and *t*.

If the *sim(s, t)* is greater than a given threshold, the two are considered semantically similar. The words *s* and *t* are relabeled to the LCS word instead; for example, "author" and "writer" are semantically similar and are thus relabeled to their LCS, "author".

For cases in which XML element names do not exist in the dictionary, edit-distance similarity is used instead to measure similarity; for example, short forms of initial page, "initPage," and end page, "endPage," do not appear in WordNet.

4.1.2.2.2 Validate leaf-node parents

Leaf-node parents are used as clustering points. They sometimes generate many clustered subtrees, which extend the time required to compare similarity.

```
DELETE FROM leafnode_parent
WHERE ppathexp IN (
  SELECT ppathexp
  FROM leafnode_parent
  GROUP BY ppathexp
  HAVING count(pathexp) = 1)
```

Figure 6. SQL for removing leaf-node parents without a one-to-one relationship

To reduce the number of subtrees generated, the concept of instance statistics [14] is applied to check the relationship between the leaf-node parents and their children. Leaf-node parents that do not have a one-to-one relationship to their children are removed from the *leafnode_parent* relation using the SQL in Figure 6.

The leaf-node parent #/SigmodRecord#/issue#/articles#/article#/authors#/author in Figure 1 is eliminated at this point. Thus, the remaining leaf-node parents from SIGMOD Record in Figure 1(a) are:

(1) #/SigmodRecord#/issue and

(2) #/SigmodRecord#/issue#/articles#/article.

The leaf-node parents from DBLP in Figure 1(b) are:

(1) #/dblp#/inproceedings and

(2) #/dblp#/proceedings.

If the "author" node is considered a leaf-node parent, it would generate at least a many subtrees as are generated by the "article" node.

4.1.3 Clustering XML documents into subtrees

The remaining leaf-node parents are used to generate appropriate subtrees containing a variety of information. The subtrees can be generated by the pseudocode shown in Figure 7.

```
$record_set = SELECT distinct e.docid, e.pathid as rootsubtree, e.st, e.ed
FROM leafnode_parent p, element1 e
WHERE p.docid = e.docid
AND p.ppathid = e.pathid
AND e.docid = $docid
ORDER BY e.docid, st

for each $r in $record_set{
   INSERT INTO subtree(docid, ppathid, pst, ped,pathid, st,ed, key, subtreeid, value)
   SELECT docid, ppathid, scope_start, scope_end, pathid, st, ed, ' ',subtreeid , value
   FROM txt1
   WHERE docid = $r.docid
   AND st >= $r.st
   AND ed <= $r.ed)
}
```

Figure 7. Pseudocode for generating subtrees

First, using the attributes start and end from the element relation, the region covered by each leaf-node parent is selected. These regions are matched with the regions in the text relation in order to identify all content nodes at the leaf-node level under the leaf-node parent. The content values for each leaf-node parent are grouped together by the attribute "subtreeid."

There are two types of clustered subtrees: simple and complex. A *simple subtree* has only leaf nodes under its root. A *complex subtrees* has one or more subtrees under its root and at least one leaf node originating from its root.

In Figure 1, the SIGMOD Record document is clustered into two subtree levels. One is rooted by the "issue" node and categorized as complex; the other one is rooted by the "article" node and cauterized as simple. No subtrees are rooted by the "author" node because that node is dismissed after the leaf-node parents are validated.

4.1.4 Key Generator

A key is a unique value that can be used to identify a particular item or distinguish items from others. The key of a subtree is modeled as an XML attribute, which is one of leaf nodes in a subtree. It has a unique value and is able to identify other attributes. Possible keys for an XML document are identified by the SQL query in Figure 8, which retrieve unique values from the text relation that can be used to distinguish among items.

```
SELECT docid, pathid, value
FROM text
GROUP BY docid, PathID,Value
HAVING Count(Value) = 1
```

Figure 8. SQL query for finding keys

The leaf nodes returned are considered keys. The value, "Y," is flagged in the attribute "key" on the matched records (according to their docid, pathid, and value) in the *subtree* relation.

From this point, same subtrees of XML documents may not contain a key. Further, an item in either a base or a target XML tree may have a key but not appear in the other XML tree. Finally, a subtree may have multiple alternate keys. Consequently, subtree matching using keys may cause one-to-multiple matching. To find the best match for this case, the remaining subtrees are compared. When available, however, a key can reduce the number of matchings.

4.1.5 Similarity Components

This procedure has two steps: (1) matching subtrees with keys and (2) matching subtrees using similarity measurements based on XML content and structure.

The keys found as described in Section 4.1.4 facilitate subtree matching and identification of the best matched subtree pairs. Measurements of subtree similarity are then used to compare the remaining unmatched subtrees.

4.1.5.1 Match with keys and analyze subtree-pair matching

First, the subtrees from the base and target documents are matched using keys generated by the key generator using the SQL in Figure 9. They are matched by comparing the leaf-node values, which are marked as "Key." The results are stored in a temporary table called *v_key_match*. This table will be used later to characterize subtree matches as either one-to-one, which is considered the best type, or one-to-multiple, which can occur due to multiple alternate keys.

4.1.5.2 Analyze results from matching with keys

The results of matched subtree pairs (one-to-one) are then stored in the subtree_similarity_score relation and flagged as "Key" in order to distinguish the match type. The matched subtree pairs categorized as one-to-multiple matching are analyzed to identify unnecessary an inappropriate leaf-node parents to be compared with the subtrees in the other document.

The subtrees generated from the leaf-node parent #/SigmodRecord#/issue in Figure 1(a) should not be compared with the subtrees in Figure 1(b) because they are different kinds of entities.

```
SELECT DISTINCT s1.docid as base_docid, s1.subtreeid AS base_subtreeid, s2.docid as
target_docid, s2.subtreeid AS target_subtreeid
FROM subtree s1, subtree s2
WHERE s1.docid = docid of the base document
AND s2.docid = docid of the target document
AND (s1.KEY = 'Y'
AND s2.KEY = 'Y')
AND s1.VALUE = s2.VALUE
```

Figure 9. SQL query for matching with keys

```
SELECT 'doc_base' as doc_type, base_docid as docid, base_subtreeid as subtreeid, count(*) as
match_cnt
FROM v_key_match
GROUP BY base_docid, base_subtreeid
HAVING count(*) > median # of alternate keys in the base document
UNION
SELECT 'doc_target' as doc_type, target_docid as docid, target_subtreeid as subtreeid, count(*)
as match_cnt
FROM v_key_match
GROUP BY target_docid, target_subtreeid
HAVING count(*) > median # of alternate keys in the target document
```

Figure 10. SQL query using key matching to find multiple matched subtrees

```
SELECT distinct docid, ppathid
FROM subtree
MINUS
SELECT distinct v.docid, s.ppathid
FROM v_key_manymatching v, subtree s
WHERE v.docid = s.docid and
v.subtreeid = s.subtreeid
```

Figure 11. SQL query for finding appropriate leaf-node parents

In order to reduce the number of one-to-multiple matches the matching information (i.e., the results from the SQL query shown in Figure 10) is analyzed to identify unnecessary leaf-node parents generating subtrees. The SQL query in Figure 11 returns only the path expression of leaf-node parents that have fewer subtree matchings than the median number of alternate keys. These leaf-node parents are considered appropriate clustering points. Thus, the clustered subtrees not rooted by these points are dropped.

Like the subtrees rooted by the "issue" node, the complex subtree should contain many alternate keys because the keys from each article subtree are part of the issue subtree. The leaf-node parent #/SigmodRecord#/issue in Figure 1(a) is dismissed. Therefore, the subtrees that must be compared now are those rooted as described below.

For SIGMOD Record in Figure 1(a):

(1) #/SigmodRecord#/issue#/articles#/article

For DBLP in Figure 1(b):

(1) #/dblp#/inproceedings and

(2) #/dblp#/proceedings

4.1.5.3 Match with XML content and structure

To determine which subtrees are an appropriate match the remaining subtrees are analyzed for both the content and structure of the base and target XML trees by comparing PCDATA value (content approach) and signatures (structure approach). Three components are used to compute the degree of similarity in content and structure: using Subtree Similarity Degree based on the base document (SSD1), Subtree Similarity Degree based on the both documents (SSD2), and Path Subtree Similarity Degree (PSSD).

4.1.5.4 Content similarity

Subtree Similarity Degree based on the base document (SSD1) is the percentage of the number of leaf nodes sharing the same PCDATA value out of the total number of leaf nodes in t_{bi}. *SSD1* can be calculated using the formula

$$SSD_1(t_{bi}, t_{tj}) = \frac{n}{n_{bi}} \times 100\%$$

where t_{bi} and t_{tj} are two subtrees from the base document and target document respectively, n is the number of leaf nodes sharing the same PCDATA value, and n_{bi} represents the total number of leaf nodes in t_{bi}. *SSD2* is the ratio of leaf-node values common to both the base and target subtrees. It can be written as

$$SSD_2(t_{bi}, t_{tj}) = \frac{2 \times n}{n_{bi} + n_{tj}} \times 100\%$$

where n_{tj} is the number of leaf nodes in the target subtree.

For scoring purposes, each common node is worth 1 point, and a common node defined as a key is worth 2 points. The *SSD1* and *SSD2* scores of all remaining subtree pairs are calculated and stored in the *subtree_similarity_score* relation. The matched pair is the subtree pair with the highest similarity score.

However, the one-to-multiple matches may occur despite these steps. To find out which subtree pair is the best match, the similarity of the signature of matched leaf-node values is measured using Path Similarity Degree (PSD).

4.1.5.5 Structural similarity

Before measuring path similarity, XML element names of the both documents are semantically transformed using LCS to ensure precise results (as described in Section 4.1.2.2.1).

Path Similarity Degree (PSD) is the ratio of common labels *N* on paths from the base and target subtrees having the same PCDATA value to the number of path elements in the base subtree:

$$PSD(i) = \frac{N}{N_{bi}} \times 100\%.$$

4.1.5.6 Similarity of content and structure

Path Subtree Similarity Degree (PSSD) is an average of Path similarity degree for t_{bi} and t_{tj}:

$$PSSD(t_{bi}, t_{tj}) = \frac{\sum_{i=1}^{k} PSD(i)}{k} \times SSD(t_{bi}, t_{tj}) \times 100\%$$

where i is the total number of matched leaf node paths in the base subtree between 1 to k paths. The matched subtree is the pair of subtrees that has the highest degree of subtree similarity above a certain threshold in terms of content and structure.

4.2 System Design

This section describes the system design, requirements, and implementation.

The system relies on the programming language Java 5.0 (JDK 5) and on Oracle 10G database. XRel uses the validating XML parser and simple API for XML (SAX) in order to convert XML documents into the relations described in Section 4.1.1, and it uses JDBC to connect with the database.

For the interface, the XML documents are parsed by the Java API for XML processing (JAXP), which enables applications to parse, transform, validate, and query XML documents using an API independent of any specific XML processor implementation. The XML documents are then displayed in a tree data structure by JTree.

5. ALGORITHM

Figure 12 illustrates the approach described in Section 4 and written in pseudocode. This algorithm is processed after XML documents are parsed into a relational database. There are three main modules: (1) subtree generator and validator, (2) key generator, and (3) subtree matching by similarity components.

The inputs of this algorithm are two XML documents stored in a relational database. First, the XML documents are fragmented into small subtrees using leaf-node parents. In Module 1, the leaf-node parents are identified and validated using the taxonomic analyzer and the instance statistics concepts. The XML documents are clustered into subtrees. Finally, subtrees are generated using the leaf-node parents.

All possible keys are identified in Module 2 and used in turn to identify their subtrees. At this point, the subtree relation is updated by marking the attribute “key” as ‘Y’ (see Section 4.1.4).

Module 3, subtree matching, is separated into two sub modules, Module 3.1 and Module 3.2.

Algorithm XDI-CSSK
Input: XML document tree T_b and T_t
Output: Set of matched subtree pairs $\{(t_{bi},t_{tj})\}$

```
//Module 1: Generate and validate subtree
Find_leafnode_parent();          //Figure 5
Validate_leafnode_parent();      //Figure 6
Generate_subtree()               //Figure 7

//Module 3: Subtree Matching
//Module 2: Identifying key(s)
Finding_key();                   //Figure 8

//Module 3.1: Matching subtrees
Match_with_key();                //Figure 9
Find_proper_lefanode_parent   //Figure 10 and Figure 11

//Module 3.2: Subtree similarity degree
for (every t_bi in T_b){ //non- matched subtrees from the base document tree
MaxSim[i] = 0;
for (t_tj in T_t){ //Subtrees from the target document tree
   CalSimilarity S(t_bi ,t_tj)      //Section 4.1.5.4
   MaxSim[i] = Max(MaxSim[i], S(t_bi ,t_tj));
}
StoreMSSD(t_bi, t_tj, MaxSim[i]);  //Store Max SSD in a temporary table
      }
      //Path Subtree Similarity degree computation
      for (every t_bi in MSSD, such that Count(MaxSim()) >1 and MaxSim > τ ){  //Count the number of maximum
similarity degrees
//Match subtree more than one pair
MaxPath[i] = 0
for (j = 1 to k_t){
   CalPathSimilarity PSSD(t_bi ,t_tj) //Section 4.1.5.5
   MaxPath[i] = Max(MaxPath[i], PSSD(t_bi ,t_tj));
}
StoreMPSSD(t_bi, t_tj, MaxPath[i]);
      }
      Return (t_bi,t_tj) stored in Max SSD and Max PSSD
```

Figure 12. XDI-CSSK Algorithm

First, the clustered subtrees from the both base and target documents are compared by the key identified in the function "match_with_key()". The results can be the best matched subtrees or multiple matched subtrees. The best matched subtrees are stored as outputs. Multiple-matched subtrees occur when the subtrees have more than one alternate key. These subtrees are not considered the best matched. The results of multiple

matched subtrees are analyzed by the function "find_proper_leafnode_parent" to find and eliminate irrelevant subtrees. This function compares the number of keys with the median number of alternate keys per subtree. These irrelevant subtrees no longer count as subtrees, so they are removed from the *subtree* relation. The remaining unmatched and multiple-matched subtrees from this module are identified to find the degree of subtree similarity in Module 3.2.

For Module 3.2, the remaining subtrees t_{bi} from the base document tree T_b are selected. Each subtree is compared with the remaining subtrees t_{tj} from the target document T_j to calculate subtree similarity degrees (*SSD1* and *SSD2*). The maximum degree of subtree similarity (MaxSim) for each subtree t_{bi} is stored in a temporary table for later identification of the best match. The best match is defined as the subtree pair (t_{bi}, t_{tj}) with the maximum degree of subtree similarity, which must be greater than a user-defined threshold τ. In the cases when we have more than one pair fit these criteria, PSSD is computed for those pairs, and the best matched subtree is that with the maximum path subtree similarity degree.

Table 1. Clustering points between XDoI and XDI-CSSK

Document	**XDoI**	**Number of subtrees**	**XDI-CSSK**	**Number of subtrees**
SIGMOD Record	#/SigmodRecord#/issue #/SigmodRecord#/issue#/articles#/article #/SigmodRecord#/issue#/articles#/article#/authors#/author	67 1504 1504	#/SigmodRecord#/issue#/articles#/article	1504
DBLP1	#/dblp#/inproceedings	769	#/dblp#/inproceedings	769
DBLP2	#/dblp#/inproceedings #/dblp#/proceedings	803 2	#/dblp#/inproceedings #/dblp#/proceedings	803 2
DBLP3	#/dblp#/inproceedings #/dblp#/proceedings	1421 17	#/dblp#/inproceedings #/dblp#/proceedings	1421 17

Table 2. The numbers of subtree comparisons required

	XDoI	XDI-CSSK
1st pair	1208099	1156576
2nd pair	1264655	1210720
3rd pair	2259098	2162752

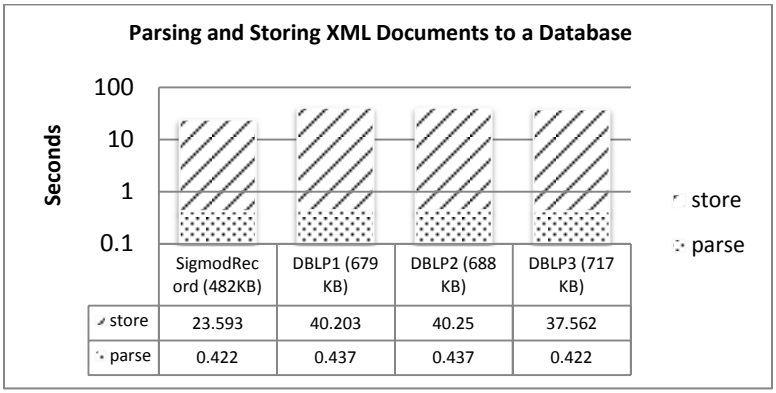
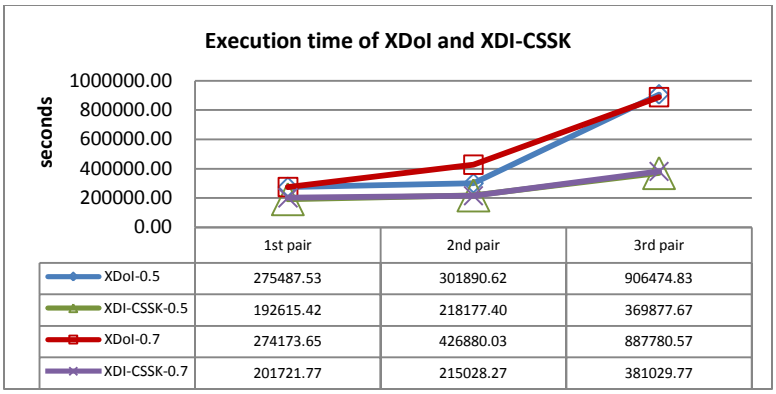


Figure 13. XRel parsing and storing XML documents

Figure 14. Quality of XDoI and XDI-CSSK results



	1st pair	2nd pair	3rd pair
XDoI-0.5	275487.53	301890.62	906474.83
XDI-CSSK-0.5	192615.42	218177.40	369877.67
XDoI-0.7	274173.65	426880.03	887780.57
XDI-CSSK-0.7	201721.77	215028.27	381029.77

Figure 15. Execution time of XDoI and XDI-CSSK

From this point, the best matched subtree pairs are the outputs. They can be joined together in order to integrate them. Joining matched subtrees is not difficult because they are stored in a relational database.

6. PERFORMANCE RESULTS

This section evaluates the improvement in XML document clustering and similarity comparison made possible by this approach. Experiments were designed to compare XML documents from SIGMOD Record [1] and DBLP [16]. They used the same data sets used for the XDoI experiments, that is SIGMOD Record and portions of 700 KBs of DBLP. Three fragments of DBLP randomly selected from the XDoI experiments were used; they are labeled named DBLP1, DBLP2, and DBLP3. The first document pair is SIGMOD Record and DBLP1, the second is SIGMOD Record and DBLP2, and the third is SIGMOD Record and DBLP3.

First, the XML documents were parsed and stored in an Oracle 10G database using XRel [17]. This process is illustrated in Figure 13 using the vertical axis with log scale; it took less than 41 seconds to complete.

Second, the difference in the clusterings between XDoI and XDI-CSSK were evaluated. Table 1 shows the clustering points and the number of clustered subtrees from

XDoI and XDI-CSSK. The number of clustered subtrees on SIGMOD Record from XDI-CSSK is less than that from XDoI. Based on validation of leaf-node parents and on the information from multiple matching using keys, the system identified unnecessary leaf-node parents. Therefore, only appropriated and relevant subtrees remained to be measured.

For both approaches, experiments were also conducted to evaluate the quality of results and the overall time required to cluster, identify keys, measure subtree similarity, and identify matches. Each approach was tested with three pairs of fragments from the two documents. The similarity thresholds for the experiments were 0.7 % and 0.5%. The quality of subtree matching result, Q was defined as Sn/An, where *Sn* is the number of subtrees matched by a given approach and *An* is the number of actual matched subtrees. As shown in Figure 14, the quality of results was identical for both approaches are identical because they use the same similarity measures. The higher the threshold, the higher the quality. Figure 15 shows the execution time for XDI-CSSK and XDoI. The former outperformed the latter because it was able to eliminate inappropriate subtrees using subtree validation and information from the results of multiple matches based on keys. In addition, identification of common content using SQL queries of indexed relations improved the computation time. The third pair in Figure 15 required more execution time because the number of clustered subtree comparisons was higher than in the first two pairs shown in Table 2.

7. CONCLUSIONS

This paper has described XDI-CSSK, a system that determines the degree of semantic similarity using XML content and structure, as well as the concept of XML keys. Major challenges in XML integration include identification of appropriate subtrees representing individual objects and identification and elimination of clustered subtrees, which are the main factors causing high computation cost in similarity measurements. Leaf-node parents were used as clustering points and validated using instance statistics and a taxonomic analyzer. The results of subtree matching based on defined keys were

also used to purge unrelated subtrees to the subtrees in other document. Matching subtrees with the keys and validating subtrees by eliminating unnecessary subtrees certainly reduces the workload of the system in terms of subtree similarity comparison. The experiments demonstrated that XDI-CSSK works effectively with the bibliographical documents, SIGMOD Record and DBLP. It will likely work just as well with other domains and types of XML trees (shallow, deep, etc.).

8. FUTURE WORK

This work applied taxonomy of concepts to determine the structural similarity (or path similarity) of XML documents from XML element names. Comparison of content for XDI-CSSK still depends on a string matching technique, which may not reveal exactly how similar the two strings are. Future work will focus on identifying the semantic similarity of content by applying the taxonomy of concepts with acceptable execution time. In addition, it will expand the scope of comparison to address two XML documents of different versions.

9. REFERENCES

[1] *ACM SIGMOD Record in XML*. (n.d.). Retrieved March 2006, from http://www.acm.org/sigmod/record/xml

[2] Augsten, N., Bohlen, M., & J, G. (2005). Approximate matching of hierarchical data using pq-grams. *Proceedings of the 31st international conference on Very large data bases* (pp. 301-312). VLDB Endowment.

[3] Bille, P. (2003). Tree edit distance, alignment distance and inclusion. *IT Univ. of Copenhagen TR-2003-23* .

[4] Buneman, P., Davidson, S., Fan, W., Hara, C., & Tan, W. (2002). Keys for XML. *Computer Networks* , 473-487.

[5] Christiane, F. (1998). *WordNet: An Electronic Lexical Database.* MA: MIT press Cambridge.

[6] Jiang, J., & Conrath, D. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. *Jiang, J.J. and Conrath, D.W.* , 19-33.

[7] Lian, W., Cheung, D. W.-l., Mamoulis, N., & Yiu, S.-M. (2004). An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *IEEE Transactions on Knowledge and Data Engineering , 16* (1), 82-96.

[8] Liang, W., & Yokota, H. (2006). A path-sequence based discrimination for subtree matching in approximate XML joins. *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, (pp. 23-28).

[9] Liang, W., & Yokota, H. (2005). LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration. *Proceedings Of BNCOD 2005*, (pp. 82-97).

[10] Liang, W., & Yokota, H. (2006). SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes. *IPSJ Digital Courier , 2*, 382-392.

[11] Lin, D. (1998). An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 296-304).

[12] Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *Proceedings of the 14th Internaltional Joint Conference on Artificial Intelligence* , 448-453.

[13] Viyanon, W., Madria, S. K., & S, B. S. (2008). XML Data Integration Based on Content and Structure Similarity Using Keys . *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems* , 484-493.

[14] Weis, M. (2005). Fuzzy Duplicate Detection on XML Data. *Proceedings of VLDB 2005 PhD Workshop*, (p. 11).

[15] Wu, Z., & Palmer, M. (1994). Verbs semantics and lexical selection. *Proceedings of the 32nd annual meeting on Association for Computational Linguistics* (pp. 133-138). Association for Computational Linguistics Morristown, NJ, USA.

[16] *XML Version of DBLP*. (n.d.). Retrieved May 2006, from http://dblp.uni-trier.de/xml/

[17] Yoshikawa, M., Amagasa, T., Shimura, T., & Uemura, S. (2001). XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology , 1* (1), 110-141.

[18] Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing , 18*, 1245.

III. XML-SIM: STRUCTURE AND CONTENT SEMANTIC SIMILARITY DETECTION USING KEYS

Waraporn Viyanon and Sanjay K. Madria

Department of Computer Science, Missouri University of Science and Technology
Rolla, Missouri, USA
{wvz7b@mst.edu, madrias@mst.edu}

Abstract. This paper describes an approach to detecting similarity in structure and content semantically between two XML documents from heterogeneous data sources using the notion of *keys*. Comparison with previous systems (XDoI and XDI-CSSK) demonstrates that this new approach performs significantly better, yields fewer false positives, and offers a shorter execution time.

Keywords: XML similarity detection, keys, clustering, matching

1. INTRODUCTION

XML has been increasing relevance as a means to exchange information and present complex data on the Internet [7]. XML sources with similar content may be described using different tag names and structures; bibliographical data such as DBLP [20] and SIGMOD Record [1] are examples. Integration of the similar XML documents from different data sources benefits users, permitting them access to more complete and useful information.

XML documents encode both structure and data. In order to integrate them, therefore, similarities in both structure and content must be accurately identified. Most matching algorithms treat XML documents as a collection of items represented in tree form. These trees are fragmented into small, independent subtrees. The subtrees can be analyzed to identify similarities of content and structure between two XML documents.

Subtree pairs with a degree of similarity above a given threshold are considered matched pairs; these can be integrated into one XML document. Recent works on XML document integration have introduced systems [3, 4, 6, and 12] such as SLAX. Other studies have shown that integration techniques such as XDoI [17] and XDI-CSSK [18] outperform SLAX. Both these approaches require that the degree of similarity in the content of XML documents must first be determined; structure is considered only later. Ideally, however, structure should be considered first, and similarities of content between two subtrees should be evaluated only for structurally similar pairs. The degree of similarity in content is measured by computing common leaf-node values from a subtree in a base XML document with those in the target document. This is a time consuming method; however, leaf-node values should be compared only when they have similar structures.

This paper describes the design and implementation of a system to integrate XML documents based on the similarity of their structure and content using keys and semantic matching. This framework is an improvement on XDoI [17] and XDI-CSSK [18]. It focuses on the semantics associated with the child nodes in a subtree, thus reducing the number of subtree comparisons to be made. The contributions of this paper can be summarized as follows:

1. It proposes an improved framework for XML integration based on previous methods that cluster XML documents into subtrees, identify and match subtrees using keys [17], use the Java WordNet similarity library (JWSL) to apply the metric of semantic similarity based on information content [14]. It defines a new method of computing the similarity between two XML documents in terms of both structure and content and it describes the implementation of an algorithm based on structure (path) semantic similarity for matching subtrees.
2. It describes experiments performed on bibliographical data sources, ACS SIGMOD Record [1] and DBLP [20], and evaluates the proposed framework by comparing it with previous systems. This comparison demonstrates a clear improvement in parameters such as similarity detection and execution time [17, 18]. It also shows that the approach presented here reduced the number of false positive by 12.84%.

2. RELATED WORK

Similarity detection in XML documents can be categorized as relating to either structural similarity alone or to similarity of both content and structure. Detection of structural similarity relies mostly on in document clustering and change detection. Similarity of both content and structural, however, is important in document integration.

Several approaches [3, 4, 6] to the identification of structural similarity in tree-based documents are based on finding the least edit distance [22] between two documents In other words, they determine how one document (T1) can be edited to transform it into a second document (T2). Other work on structural similarity aims to extract pure structural information from documents. Tree edit distance (TED) measures the minimum number of node insertions, deletions, and updates required to convert one tree into another. By default, TED assigns a cost value of 1 to each edit operation [3, 4]. The edit distance between two trees is the smallest cost of transforming one document into another. Tree edit distance is expressed in term of $O(n^2 min^2(l,d))$ time and $O(n^2)$ space, where n represents the number of nodes, l represents the number of leaves, and d is the depth [22].

A path is defined as a list of connected nodes starting at the root and terminating at a leaf node. Path similarity can be measured in several different ways: Binary measurement determines whether a pair of paths is equivalent; Partial measurement determines the number of comparable nodes in two paths. Finally, weighted measurement weights the nodes according to their distance from the root. Partial path similarity measures are expensive to compute because there are n-factorial mappings between the paths of two trees. They depend on exhaustive algorithms that yield an optimal similarity score.

XML DTD can evaluate similarity by comparing document type definition (DTD) of one document with that of another; however, the XML DTD may not always be available.

There are also many effective and widely used methods to detect similarity between two elements; these include string matching, edit distance, and semantic similarity. String matching determines whether strings are identical. This method is simple to implement, but it often fails to identify similar strings. The distance between

strings s and t is equal to the computational cost of the sequence of edit operations that converts s to t. As mentioned above, edit distance is time-consuming, and the results may not be semantically accurate. Another approach that is similar in many ways to edit distance is the longest common subsequence (LCS) approach [2], which finds the longest sequence of tokens common to the two strings.

Methods of identifying semantic similarity [9, 10, 13, 14, and 16] have been introduced in order to capture meaning of words. Generally, these methods can be categorized into two main groups: those based on edge counting [15] and those based on information corpus.

The information-theory-based method of identifying semantic similarity was first proposed by Resnik [16]. The similarity of two concepts is defined as the maximum probability score of the concept that subsumes them in the taxonomic hierarchy. The information content of a concept depends on the probability of encountering an instance of the concept in a corpus. This probability is determined by the frequency with which the concept and its sub-concept occur in the corpus. The information content is thus defined as the negative of the log of the probability. Jiang and Conrath [9] proposed a combined method derived from the edge-based notion by adding the information content as a decision factor. They consider the fact that edges in the taxonomy may have unequal link strength; therefore, the link strength of an edge between two adjacent nodes is determined by local density, node depth, information content, and link type. The similarity between two words is simply the summation of edge weights along the shortest path linking two words. Lin [13] derived a measure similar to Resnik's information content, but better [16]. His contribution consisted of normalizing by the combined information content of the concepts to be compared and assuming their independence.

The best known resource on taxonomic hierarchy is WordNet [8], a utility program that allows a user to compute information content values from the Brown Corpus, the Penn Treebank, the British National Corpus, or any given corpus of raw text. Pirror and Seco [14] have developed JWSL, which provides methods based on information theoretic theories of similarity.

Keys are fundamental to data models and conceptual design, and they facilitate subtree matching [5]. If keys could be identified in XML documents, the number of

matchings could be dramatically reduced. Since most XML data is data-centric (i.e., derived from the relational data model), keys can more accurately identify matching subtrees.

3. PROBLEM STATEMENT

This paper explains the drawbacks of XDoI [18] and XDI-CSSK [17]. These methods compute subtree similarity based on the similarity of content by comparing the number of common values at the leaf-node levels; they do not consider document structure. If evaluation of content similarity yields multiple matches, then the structural or path similarity is taken into account to identify the most similar subtree pair. This approach makes the computation time consuming because it requires comparison of all leaf nodes regardless of the degree of similarity in data type and semantic.

Figure 1 provides an example of the structure of SIGMOD Record and DBLP documents. To integrate these two XML documents, XDoI and XDI-CSSK cluster them into smaller subtrees using leaf-node parents as clustering points; they then compare all subtree pairs. According to previous work [18], clustering in XDI-CSSK is better than that in XDoI because the former is able to segment XML documents into proper subtrees. In the example shown in Figure 1, the clustering points are the edges above the *article* node from SIGMOD Record, and the *inproceedings* and *proceedings* nodes are from DBLP. Even XDI-CSSK clusters subtrees in order to compare those of two XML documents. The clustering process removes inappropriate subtree levels from the results of multiple matchings using keys, but in evaluating the similarity of subtrees, it considers only content, ignoring the structure of the document. The algorithms of both approaches for identifying similar content and structure are straightforward; they compare leaf nodes that share the same PCDATA value. In this example, all the leaf nodes rooted by the *article* node are compared with those rooted by the *inproceedings* and *proceedings* nodes. It would make no sense to compare the value at the *title* node in the *article* subtree with that at the *pages* or *year* node in the *inproceedings* subtree because these are not similar in terms of semantics and data type.

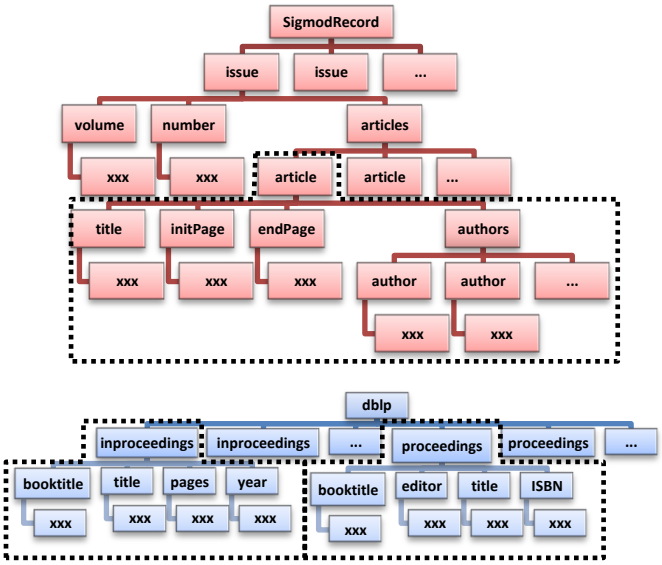


Figure 1. Example of XML documents compared in XDoI and XDI-CSSK

XDoI shows that identification of a key can reduce the number of subtree matchings, and XDI-CSSK takes advantage of the results of key matchings to eliminate inappropriate subtrees.

This paper addresses these drawbacks by considering the structural semantic similarity of leaf nodes in clustered subtrees before comparing the content of leaf nodes.

4. APPROACH

This section describes an XML document integration called XML-SIM, which detects similarities in two XML documents more effectively than either XDoI or XDI-CSSK. First, it describes the overall framework of this approach, then it provides the details of each component. Finally, it presents the algorithm for this approach.

4.1 XML-SIM Framework

XML-SIM framework consists of four components: (1) XML document storage, (2) subtree generation, (3) key generation and matching, and (4) similarity detection and subtree matching. Figure 2 illustrates this framework.

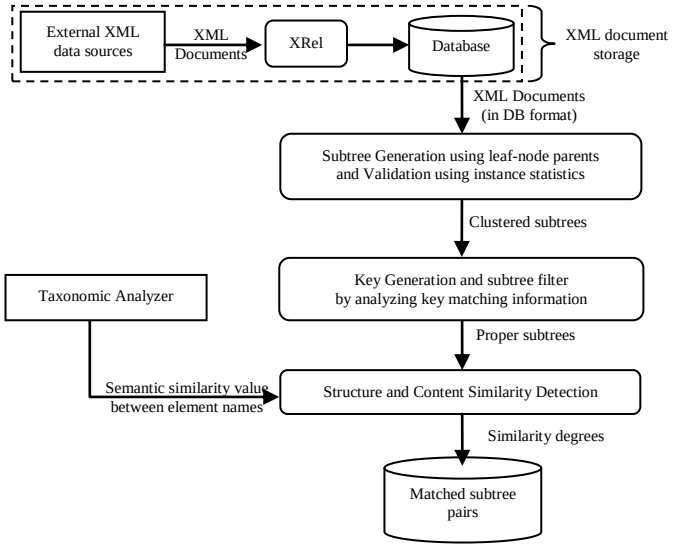


Figure 2. XML-SIM framework

First, XML documents are stored in a relational database, which increases scalability so that very large XML trees do not exceed the limits of the system. Second, XML documents are clustered into subtrees using leaf-node parents. The subtrees are verified for integrity using the concept of instance statistics [19]. XML key(s) are defined based on a leaf-node value match for all unique node values sharing the same path signature. The key are later matched with subtrees; however, key matching may result in multiple matches because a key may be a part of multiple subtrees. According to Definition 6 in Section 4.2 this key-matching information can be used in the subtree filter process to eliminate inappropriate subtrees. At this point, only appropriate subtrees remain to be compared. The structures of these subtrees are measured to find the structural semantic similarity using the taxonomic analyzer. This similarity facilitates

comparison of content. Similarity in content is determined by comparing leaf-node values with a similar semantic structure. Finally, the system identifies the best matched subtree pairs, which can then be integrated.

4.2 Key Definitions for XML-SIM

This section presents the notations, definitions, and algorithm that solve the problem described in Section 3 above.

4.2.1 XML Document Storage

Following are some definitions of terms related to XML documents and a description of the storage model.

Definition 1. XML document tree: *An XML document tree T_i is an ordered labeled tree generated after parsing an XML document. T_i denoted as $T_i = (V, v_0, E)$ where V is the set of nodes; v_0 is the root node; E* is *the set of edges in the tree T_i. T_b is a base document tree, and T_t is a target document tree.*

The XML documents are loaded into a relational database using XRel [21], which decomposes the document into nodes on the basis of its tree structure and stores it in relational tables according to the node type, with information on the path from the root to each node. XRel consists of the four relational schemas shown in Figure 3.

Element(docID, pathID, start, end, index, reindex)
Attribute(docID, pathID, start, end, value)
Text(docID, pathID, start, end, value)
Path(pathID, pathexp)

Figure 3. XRel schemas

The database attributes “docID,” “pathID,” “start,” “end,” and “value” represent the document identifier, simple path expression identifier, start position of a region, end position of a region, and string value, respectively. Element nodes or leaf nodes are identified by their region and stored in the relations *Element* and *Text*. To identify each of the attribute nodes, the attribute name is retained as the suffix of the simple path

expression of an attribute node and the attribute value is stored in the relation *Attribute*. The database attribute "pathexp" in the relation Path stores simple path expressions as explained in Definition 2.

Definition 2. Path expression: *Any node v_i can be identified by its location within a tree T_i by a path expression or signature p_i. A path expression p_i consists of one node or nodes from the node set V separated by "/".* In Figure 1, the path expression of the node *title* is as */sigmodRecord/issue/articles/article/title*. The path expressions are used to measure structural semantic similarity.

4.2.2 Subtree Generation

Below are definitions related to subtree clustering, followed by a discussion of the subtree generation phase.

Definition 3. Leaf-node parent: *For a document tree T_i with a node set V and an edge set E, v_p is a leaf-node parent,* if *(1) $v_p \in V$ (2) $(v_l, v_p) \in E$, where v_p is the parent of v_l, and v_l is a leaf node.*

In other words, a leaf-node parent is a node that has at least one child leaf node. This leaf-node parent is considered a subtree root in the clustering process. In Figure 1, the leaf-node parents are the nodes, "issue," "article," and "author." They can be found using the SQL query in Figure 4.

```
SELECT distinct docid, p.pathid as pathid, pathexp
FROM text l, path p
WHERE p.pathid = l.pathid
```

Figure 4. SQL query for finding leaf-node parents

Definition 4. Clustering point: *An edge e_c lies between nodes v_p and v_i. This edge is a clustering point iff $(v_p, v_i) \in E$, where v_i is the parent of v_p, and v_p is a leaf-node parent (as described in Definition2). The edge e_c is deleted to generate a subtree t_i denoted as $t_i = (V_i, v_p, E_i)$. The clustering point is the point at which an XML tree is*

clustered into subtrees. The clustered subtrees are categorized as either, *simple* or *complex:*

Definition 5. Simple subtree: *Two XML document trees T_b (the base tree) and T_t (the target tree.) are clustered into k_b and k_t subtrees respectively, where t_{bi} is a node in T_b such that $1 \leq i \leq k_b$ and t_{tj} is a node in T_t such that $1 \leq j \leq k_t$. The subtree t_{bi} with a node set V_{bi} is simple subtree if (i) $num\_parent(v_{pi})$ is equal to 0, (ii) $num\_parent(v_{pi})$ is equal to 1, the parent of $v_{pi}$is NOT a leaf-node parent (see Definition 3), v_{pi} is a leaf-node parent in the subtree t_{bi}, and num_parent() is a function that counts the number of parents. This condition applies to the subtree t_{tj} as well. A simple subtree is a clustered tree with only a root and one or more of leaf nodes*

Definition 6. Complex subtree: *Any clustered subtrees $t_i^{'}$ with a node set $V_i^{'}$ is complex if the parent of $V_{pi}^{'}$ is a leaf-node parent. A complex subtree is a clustered subtree with at least one simple subtree, a root, and one or more of leaf nodes.*

The leaf-node parent and clustered subtrees are also stored in the relational database, as shown in Figure 5. The *leafnode_parent* relation stores path signatures that have leaf-nodes and their parent path expressions. The pathids of parent paths can be retrieved from the *path* relation. The *subtree* relation stores the clustered subtrees, which are used later to compare subtree similarity. Each subtree contains path information, content values at the leaf-node level, and a key flag. The key flag is used to identify multiple unique leaf nodes with the same pathid. Key generation is discussed in Section 4.2.4.

Leanode_parent(docID, ppathExp, ppathid, pathexp, pathid)
Subtree(docID, ppathID, pst, ped,pathid, st, ed, value, key, subtreeid)

Figure 5. XML-SIM relations

4.2.3 Subtree Validation

A subtree representing an independent object should contain nodes representing various types of information, rather than just one kind of node. For example, in Figure 1 <authors> is the parent of two <author> nodes, which are its leaf nodes. The <authors> node is considered the root of the subtree that has two <author> nodes as its children. Clearly, this kind of subtree contains no information other than <author>; therefore, extraction of such a subtree is not useful for evaluating subtree similarity.

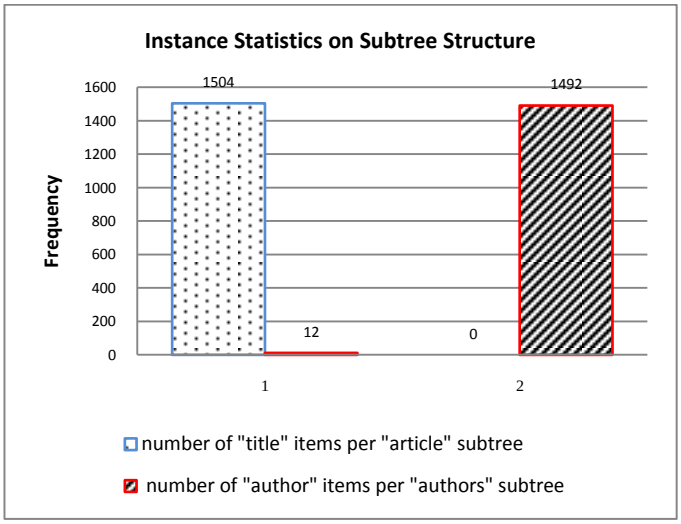


Figure 6. Instance statistics on subtree structure

The concept of instance statistics based on the element structure of subtrees [19] is applied to check the relationship between the elements of the leaf-node parent and those of its children leaf-node elements. This process determines whether they preserve a loose one-to-one relationship by capturing how often an instance (or a subtree) of leaf-node parents includes a particular number of instances of children. Figure 6 shows that there is only one <title> element per <article> subtree; on the other hand, most “authors” subtrees have two “author” elements. The relationship between <article> and <title> is, therefore, one to one. The leaf-node parents that do not have a loose one-to-one

relationship with their children are removed from the *leafnode_parent* relation using the SQL in Figure 7.

```
DELETE FROM leafnode_parent
WHERE ppathexp IN (
   SELECT ppathexp
   FROM leafnode_parent
   GROUP BY ppathexp
   HAVING count(pathexp) = 1)
```

Figure 7. SQL query to remove leaf-node parents lacking a loose one-to-one relationship

4.2.4 Key Generation

The key of a subtree is modeled as an XML attribute, which is one of leaf nodes in a subtree. It has a unique value and is able to identify other attributes in its subtree. Possible keys for the XML documents are identified by the SQL query in Figure 8. This query retrieves unique values from the *text* relation that can be used to distinguish among various items.

```
SELECT docid, pathid, value
FROM text
GROUP BY docid, PathID,Value
HAVING Count(Value) = 1
```

Figure 8. SQL query to identify leaf nodes as keys

Definition 7. Subtree key: *A subtree key is a leaf node v_k that has a unique value. This leaf node is compared with any leaf nodes v_l* having *the same path expression p_k of the node v_k.*

The query in Figure 8 returns leaf nodes, the labels of which are considered subtree keys. In the subtree relation "Y" in the attribute "key" on the matched records (according to their docid, pathid, and value) is flagged.

4.2.5 Subtree Matching Using Subtree Keys

The subtree keys thus identified are used to match subtrees. Those subtrees whose leaf nodes (labels) are marked as "key" and have identical values are compared. The key matching results are stored in a temporary relation called *v_key_match*. Subtree key matching may cause multiple matchings, stored in *v_key_manymatching*, because complex subtrees contain multiple subtrees that may have leaf nodes defined as keys. Although this comparison disregards the structure of the leaf nodes, its matching results can be analyzed to identify any subtree level inappropriate for comparison. The matching information is analyzed by determining the difference between the number of subtree matches and the median number of alternate keys. This step assumes that a complex subtree may contain a huge number of simple subtrees, which in turn contain alternate keys. Such complex subtrees yield inappropriate. To eliminate inappropriate subtrees, a threshold is calculated using the median number of alternate keys. Subtrees that produce more multiple matches more than the median number of alternate keys are eliminated. The results of the key matching are retrieved by the SQL query in Figure 9, and the SQL queries in Figure 10 identify appropriate leaf-node parents.

At this point, we have filtered the subtrees and got the appropriate subtrees from both XML documents to be compared in the structure and content similarity detection.

```
SELECT DISTINCT s1.docid as base_docid, s1.subtreeid AS base_subtreeid,
s2.docid as target_docid, s2.subtreeid AS target_subtreeid
FROM subtree s1, subtree s2
WHERE s1.docid = docid of the base document
AND s2.docid = docid of the target document
AND (s1.KEY = 'Y'
AND s2.KEY = 'Y')
AND s1.VALUE = s2.VALUE
```

Figure 9. SQL query for key matching

```
Part (a)
SELECT 'doc_base' as doc_type, base_docid as docid, base_subtreeid as
subtreeid, count(*) as match_cnt
FROM v_key_match
GROUP BY base_docid, base_subtreeid
HAVING count(*) > median # of alternate keys in the base document
UNION
SELECT 'doc_target' as doc_type, target_docid as docid, target_subtreeid as
subtreeid, count(*) as match_cnt
FROM v_key_match
GROUP BY target_docid, target_subtreeid
HAVING count(*) > median # of alternate keys in the target document

Part (b)
SELECT distinct docid, ppathid
FROM subtree
MINUS
SELECT distinct v.docid, s.ppathid
FROM v_key_manymatching v, subtree s
WHERE v.docid = s.docid and
v.subtreeid = s.subtreeid
```

Figure 10. Filtering subtrees: (a) SQL query to find multiple matches beyond the median number of alternate keys and (b) SQL query to find appropriate leaf-node parents

4.2.6 Detection of Similarity in Structure and Content

To detect appropriate matched subtree pairs, both the structure and content of the base and target XML trees must be considered. First, the degree of semantic similarity between paths must be determined based on path signatures.

Notation. For any subtree $t_i = (V_i, v_p, E_i)$ rooted by distinct labels of node v_p, let $V_l = \{v_{l1}, v_{l2}, \dots, v_{ln}\}$ be a collection of leaf nodes in t_i iff $V_l \in V_i$. Consider $P_l = \{p_{l1}, p_{l2}, \dots, p_{ln}\}$ as a collection of path expressions (defined in Definition 2) of the leaf nodes in V_l which has n elements.

All p_{lj} in the base subtree where $1 \leq j \leq n_b$ are compared with all p_{lk} in the target subtree where $1 \leq k \leq n_t$. The terms n_b and n_t represent the number of leaf nodes in the base subtree and target subtree respectively to determine the semantic

similarity of the paths. To measure the similarity between p_{lj} and p_{lk}, the node labels of both must first be compared.

Definition 8. Node label semantic similarity degree (NSSD): For each pair of path expressions p_{lj} and p_{lk}, let $V_j = \{v_{j1}, v_{j2}, \dots, v_{jn_b}\}$ and $V_k = \{v_{k1}, v_{k2}, \dots, v_{kn_t}\}$ denote a series of nodes in p_{lj} and p_{lk} respectively. The node label semantic similarity degree is based on methods of Jiang and Resnik [9, 16] and defined as

$$NSSD\left(l(v_j), l(v_k)\right) = 1 - \frac{IC\left(l(v_j)\right) + IC(l(v_k)) - 2sim(l(v_j), l(v_k))}{2}. \quad (1)$$

The IC value is calculated by considering the negative log of the probability:

$$IC(c) = -log\ p(c) \quad (2)$$

where $p(c)$ is the probability of having c in a given corpus and c is a concept in WordNet. The use of the negative likelihood is based on the notion that the more likely the appearance of a concept, the less information it coveys.

The function $sim(c_1, c_2)$ is evaluated by using their subsumer $S(c_1, c_2)$ of c_1, c_2:

$$sim(c_1, c_2) = max_{c \in S(c_1, c_2)} IC(c). \quad (3)$$

Definition 9. Path semantic similarity degree (PSSD): A path semantic similarity degree is expressed by the ratio of the sum the average NSSD for each node v_j in the path expression $p_{lj}$and the number of nodes in the path expression series. It can be expressed as:

$$PSSD\left(p_{lj}, p_{lk}\right) = \frac{\sum_{j=1}^{n_b} avg(NSSD_j)}{n_b} \quad (4)$$

where $avg(NSSD_j)$ is computed from:

$$avg\left(NSSD_j\right) = \frac{\sum_{k=1}^{n_t} (NSSD(l\left(v_j\right), l(v_k))}{n_t}. \quad (5)$$

Definition 10. Matched path pair (MPP): A matched path pair is the pair with the highest PSSD value:

$$MPP(p_{lj}, p_{lk}) = max_{1 \le j \le n_b; 1 \le k \le n_t} (PSSD(p_{lj}, p_{lk})). \quad (6)$$

Definition 11. Selected Path Pair: The selected path pair is the path expression with an MPP value greater that a given threshold τ_p.

The values of PSSD are stored into a *PathSim* table. The SQL query in Figure 11 retrieves the matched path pair.

```
select base_docid, base_ppathid, base_pathid, target_docid, target_ppathid, target_pathid, pathsim
from pathsim p,
(
select b_docid, b_ppathid, t_docid, t_ppathid, t_pathid, max(max_pathsim) as max_pathsim
from (
  select p.base_docid as b_docid, p.base_ppathid as b_ppathid, p.base_pathid as b_pathid,
p.target_docid as t_docid, p.target_ppathid as t_ppathid, p.target_pathid as t_pathid, max(p.pathsim) as
max_pathsim
  from pathsim p,(
          select base_docid, base_ppathid, base_pathid, target_docid, target_ppathid, max(pathsim) as
max_pathsim
          from pathsim
          group by  base_docid, base_ppathid, base_pathid, target_docid, target_ppathid
          ) max --one to many relationship may occur
    where p.base_docid = max.base_docid
    and p.base_ppathid = max.base_ppathid
    and p.base_pathid  = max.base_pathid
    and p.target_docid = max.target_docid
    and p.target_ppathid = max.target_ppathid
    and p.pathsim = max.max_pathsim
     group by p.base_docid, p.base_ppathid, p.base_pathid, p.target_docid, p.target_ppathid,
p.target_pathid
    )
  group by  b_docid, b_ppathid, t_docid, t_ppathid, t_pathid
)max -- one to one relationship
where p.base_docid = max.b_docid
and p.base_ppathid = max.b_ppathid
and p.target_docid = max.t_docid
and p.target_ppathid = max.t_ppathid
and p.target_pathid = max.t_pathid
and p.pathsim = max.max_pathsim
order by base_ppathid, target_ppathid
```

Figure 11. SQL query to identify matched path pairs

At this point, all path expressions at the leaf-node levels have been evaluated and selected. The selected paths (see Definition 11) will be used to determine the similarity in subtree content.

The following example illustrates path pair selection. Figure 1 compares the subtree rooted by the <article> node and the subtree rooted by the <proceedings> node. Table 1 shows the path expressions from both subtrees.

Table 1. Path expressions of the subtrees rooted by <article> and <proceedings>

Path expressions (p_b) in the subtree <article>	Path expressions (p_t) in the subtree <proceedings>
p_{b1} = /article/title p_{b2} = /article/initPage p_{b3} = /article/endPage p_{b4} = /article/authors/author	p_{t1} = /proceedings/booktitle p_{t2} = /proceedings/editor p_{t3} = /proceedings/title p_{t4} = /proceedings/ISBN

The node labels from both subtrees (<article>, <title>, <initPage>, <endPage>, <authors>, <author> and <proceedings>, <booktitle>, <editor>, <title>, <ISBN>) are then distinguished so that NSSD may be computed. Table 2 shows the results. Because <authors> is the plural form of <author>, is treated as the same label.

Table 2. Results of Node Label Semantic Similarity Degree (NSSD)

$NSSD\left(l(v_j), l(v_k)\right)$					
$l(v_k) \backslash l(v_j)$	article	title	initPage	endPage	author
proceedings	0.409435	0.385556	0.149673	0.281467	0.000000
booktitle	0.743695	0.840329	0.285693	0.441001	0.281880
editor	0.497065	0.503894	0.420978	0.5198375	0.587105
title	0.649263	1.000000	0.181844	0.282675	0.000000
ISBN	0.000000	0.000000	0.000000	0.000000	0.000000

Next, PSSD is calculated for each pair of path expressions:

$$avg(NSSD_{1<article>}) = \frac{NSSD(<article>,<proceedings>) + NSSD(<article>,<title>)}{2}$$

$$= 0.529349$$

and

$$avg(NSSD_{2<title>}) = \frac{NSSD(<title>,<proceedings>) + NSSD(<title>,<title>)}{2}$$

$$= 0.692778,$$

thus

$$PSSD(p_{b1}, p_{t3}) = \frac{avg(NSSD_{1<article>}) + avg(NSSD_{2<title>})}{2}$$

$$= 0.611064.$$

The same calculation is preformed for all pairs of path expressions. Table 3 shows the results and the selected path pair, (p_{b1}, p_{t3}) or (/article/title, /proceedings/title). This pair will be used to compare content. Selection of multiple path pair is possible.

Table 3. Results of Matched Path Pair (MPP)

$PSSD(p_{lj}, p_{lk})$					$MPP(p_{lj}, p_{lk})$
	p_{t1}	p_{t2}	p_{t3}	p_{t4}	
p_{b1}	0.594754	0.448988	**0.611064**	0.198748	**0.611064**
p_{b2}	0.397124	0.369287	0.347553	0.139777	0.397124
p_{b3}	0.468900	0.426951	0.40571	0.172726	0.468900
p_{b4}	0.358752	0.373401	0.264674	0.102358	0.373401
$MPP(p_{lj}, p_{lk})$	**0.594754**	0.448988	**0.611064**	0.198748	

Definition 12. Subtree similarity based on structure and content*: The PCDATA value of each subtree* $t_{bi}(1 \le i \le k_b)$ *(content approach) is compared with those of the subtrees* $t_{tj}(1 \le j \le k_t)$ *based on the selected path (structure approach) to identify the proper matched subtree pair (MSP).*

Such a comparison based on content and structure can be done simply using loops, but this method is time consuming if there are many subtrees. Instead of loops, the approach introduced here uses an SQL query to retrieve subtree pairs, a much faster

process. The subtree pairs thus identified, which are based on the same leaf-node parent, intersect to find the subtree pair that best satisfies the conditions, which have the same PC data content and a similar structure. Figure 12 presents the algorithm of identifying matched subtree pairs.

Algorithm for Definition 12
Input: set of matched path expression pairs $\{(p_b, p_t)\}$
Output: set of pairs of matched subtrees
//find matched subtree pair based on (p_b, p_t)
for each path expression pair (p_b, p_t)
{
$sp[]$ = Retrieve subtree pair (t_b, t_t) having the same PC data content on the similar path expression of (p_b, p_t) } // find matched subtree pairs based on v_p
for each v_p in p_b
{
for s_i in $sp[]$
$MSP = MSP \cap s_i$ //MSP is a set of Matched Subtree Pairs
}

Figure 12. Algorithm for retrieving matched subtree pairs

5. XML-SIM EXPERIMENT

To evaluate the efficiency and effectiveness of the XML-SIM algorithm, experiments were designed to compare it with the XDoI and XDI-CSSK algorithms in terms of accuracy and execution time.

5.1 Experimental Setup

The experiments used on Intel Core 2 Duo 2.20GHz CPU processor with 4GB of RAM running on Windows XP Professional with Sun JDK 1.6.0_02 and Oracle Database 10g Standard Edition. The bibliographical data set SIGMOD Record (482 KB) was the base document, and three segmented DBLP documents, 700 KB each, were the target documents.

Table 4. Data set information and actual matched subtree pairs

Pair	Base XML document (size KB)	Target XML document (size KB)	Actual matched subtree pairs
#1	SIGMOD Record (482)	DBLP1 (679)	343
#2	SIGMOD Record	DBLP2 (688)	321
#3	SIGMOD Record	DBLP3 (717)	67

The actual matched subtree pairs were detected manually; they are shown in Table 4. These numbers were used to identify the false positives yielded by each algorithm.

5.2 Experimental Results

This section describes the results of the experiments to compare clustering methods based on execution time and accuracy of similarity detection.

5.2.1 Evaluation of clustering method

To verify the effectiveness of clustering XML documents into subtrees, the clustering points and the number of clustered subtrees are shown in Table 5 for each algorithm. In XDoI, SIGMOD Record is clustered into three different levels, <issue>, <article>, and <authors> because the clustering method applies leaf-node parents directly without any filters. XDI-CSSK and XML-SIM employ the same concept using leaf-node parents, and they filter the clustered subtrees using instance statistics and information from key matching. For the fragmented DBLP documents, there was no difference among these three approaches because the structure of DBLP documents is shallow with only one level defined as the clustering point. Table 5 shows the results of clustering points and the number of clustered subtrees.

5.2.2 Evaluation of execution time

Experiments to determine how fast each algorithm identifies matching subtrees on each document pair were run using a threshold of value 0.5. In XDoI and XDI-CSSK, the threshold is used to measure the similarity of content but in XDI-SIM it is used to evaluate structural similarity.

Figure 13 shows the execution time for each approach in a base-10 logarithmic scale; it indicates that XDI-CSSK performs better than XDoI because it eliminates inappropriate subtrees using the key matching results. XML-SIM dramatically outperforms both earlier approaches, suggesting that comparison of structure in the early stage helps the system detect subtree similarity faster. Computation of similarity in the third pair took much more time than others because that pair had many more subtrees than other pairs.

Table 5. Results: (a) the number of clustered subtrees based on the clustering points in SIGMOD Record.xml (b) the number of clustered subtrees based on the clustering points in DBLP1, DBLP2, and DBLP3

(a)

	Clustering points	Number of clustered subtrees
XDoI	#/SigmodRecord#/issue #/SigmodRecord#/issue#/articles#/article #/SigmodRecord#/issue#/articles#/article#/authors	67 1504 1504
XDI-CSSK	#/SigmodRecord#/issue#/articles#/article	1504
XML-SIM	#/SigmodRecord#/issue#/articles#/article	1504

(b)

	Clustering points in XDoI, XDI-CSSK, XML-SIM	Number of clustered subtrees
DBLP1	#/dblp#/inproceedings	769
DBLP2	#/dblp#/inproceedings #/dblp#/proceedings	803 2
DBLP3	#/dblp#/inproceedings #/dblp#/proceedings	1421 17

5.2.3 Evaluation of Similarity Detection

The effectiveness of the new approach was evaluated by determining the number of false positives and true positives it yielded. The false positive value is the ratio of the

number of incorrectly matched subtrees to the number of actual matched subtrees; a true positive value is the ratio of the number of correctly matched subtrees to the number of actual matched subtrees. The results show that XML-SIM outperforms XDI-CSSK [18] and XDoI [17], yielding no false positives among the three pairs of documents as shown in Figure14. This accuracy is possible because the semantic structural similarity is detected at an early stage. The results of path pair selection can also identify the matching structures in the two documents.

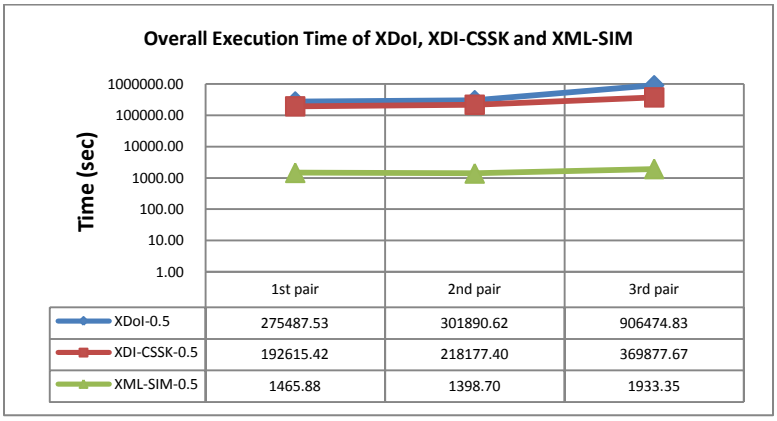


Figure 13. Overall execution time in XDoI, XDI-CSSK, and XML-SIM

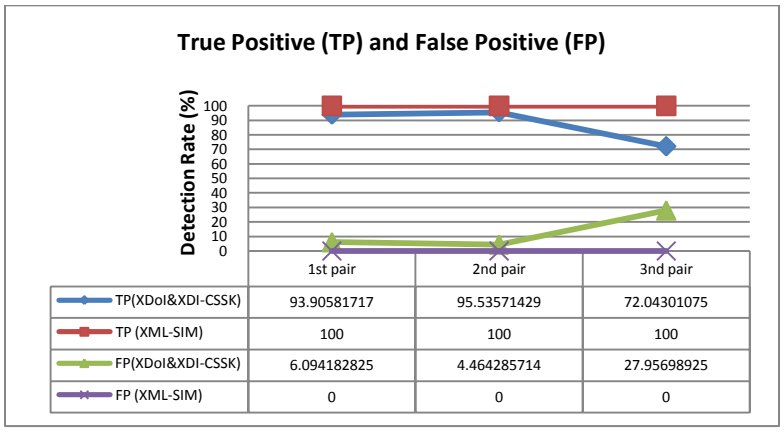


Figure 14: Detection of true positive (TPs) and false positives (FPs)

6. CONCLUSIONS AND FUTURE WORK

This paper presents an improved algorithm, XML-SIM, based on XDoI and XDI-CSSK to detect the semantic similarity between XML documents based on structure and content. This approach succeeds by determining similarity in content based on structural similarity, which is determined in turn using semantics. Experimental evaluations show that this approach outperforms XDoI and XDI-CSSK in terms of both execution time and false positive rates. Future work will seek to identify similarity among multiple versions of XML documents.

7. REFERENCES

[1] *ACM SIGMOD Record in XML.* http://www.acm.org/sigmod/record/xml (accessed March 2006).

[2] Apostolico, A., and Z. Galil. *Pattern matching algorithms.* Oxford University Press, USA, 1997.

[3] Augsten, N, M Bohlen, and Gamper J. "Approximate matching of hierarchical data using pq-grams." *Proceedings of the 31st international conference on Very large data bases.* VLDB Endowment, 2005. 301-312.

[4] Bille, P. "Tree edit distance, alignment distance and inclusion." *IT Univ. of Copenhagen TR-2003-23* (Citeseer), 2003.

[5] Buneman, P., S. Davidson, W. Fan, C. Hara, and W.C. Tan. "Keys for XML." *Computer Networks* (Elsevier) 39, no. 5 (2002): 473-487.

[6] Cobena, G., S. Abiteboul, A. Marian, and R. INRIA. "Detecting changes in XML documents." *Proceedings. 18th International Conference on Data Engineering, 2002.* 2002. 41-52.

[7] *Extensible Markup Language (XML).* http://www.w3.org/XML/ (accessed March 2006).

[8] Fellbaum, C., and others. *WordNet: An electronic lexical database.* MIT press Cambridge, MA, 1998.

[9] Jiang, JJ, and DW Conrath. "Semantic similarity based on corpus statistics and lexical ontology." *Proc. of Int. Conf. Research on Comp. Linguistics X, Taiwan.* 1997.

[10] Li, Y., ZA Bandar, and D. McLean. "An approach for measuring semantic similarity between words using multiple information sources." *IEEE Transactions on knowledge and data engineering* 15, no. 4 (2003): 871-882.

[11] Liang, W., and H. Yokota. "A path-sequence based discrimination for subtree matching in approximate XML joins." In *Proceedings. 22nd International Conference on Data Engineering Workshops, 2006.*, 23-28. 2006.

[12] Liang, W., and H. Yokota. "SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes." *IPSJ Digital Courier* (J-STAGE), 2006: 382-392.

[13] Lin, D. "An information-theoretic definition of similarity. "*Proceedings of the Fifteenth International Conference on Machine Learning.* 1998. 296-304.

[14] Pirro, G., and N. Seco. "Design, Implementation and Evaluation of a New Semantic Similarity Metric Combining Features and Intrinsic Information Content." *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems.* Springer, 2008. 1271-1288.

[15] Rada, R., H. Mili, E. Bicknell, and M. Blettner. "Development and application of a metric on semantic nets." *IEEE transactions on systems, man and cybernetics* 19, no. 1 (1989): 17-30.

[16] Resnik, P. "Using information content to evaluate semantic similarity in a taxonomy." *Proc. of IJCAI*, 1995: 448–453.

[17] Viyanon, W., and S.K. Madria. *Technical report: XDI-CSSK, A System for Detecting XML Similarity on content and structure using relational database.* Technical Report, Dept of Computer Science, Missouri University of Science and Technology, 2009 (accepted for ACM CIKM 2009).

[18] Viyanon, W., S.K. Madria, and S.S. Bhowmick. "XML Data Integration Based on Content and Structure Similarity Using Keys." *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, 2008.*. Springer, 2008. 484-493.

[19] Weis, M. "Fuzzy Duplicate Detection on XML Data." *Proceedings of VLDB 2005 PhD Workshop.* 2005. 11.

[20] *XML Version of DBLP*. http://dblp.uni-trier.de/xml/ (accessed May 2006).

[21] Yoshikawa, M, T Amagasa, T Shimura, and S Uemura. "XRel: a path-based approach to storage and retrieval of XML documents using relational databases." *ACM Transactions on Internet Technology* 1, no. 1 (2001): 110-141.

[22] Zhang, K, and D Shasha. "Simple fast algorithms for the editing distance between trees and related problems." *SIAM journal on computing* 18 (1989): 1245.

IV. XML-SIM-CHANGE: Structure and Content Semantic Similarity Detection among XML Document Versions

Waraporn Viyanon and Sanjay K. Madria

Department of Computer Science

Missouri University of Science and Technology

Rolla, Missouri, USA

{wvz7b@mst.edu, madrias@mst.edu}

Abstract. XML documents from different sources may contain the same or similar information with respect to content and structure. Query systems and search engine demand that XML documents be integrated; however, the information contained in such documents changes periodically. Therefore, it is important changes from one version of an XML document to another be detectable. Information on changes can then be used to identify semantic similarity among XML documents. This paper introduces an approach to detect similarity between XML documents that uses the change detection mechanism to join XML document versions. In this approach, subtree keys play an important role, reducing the number of unnecessary subtree comparisons among different XML versions of the same document. It uses a relational database to store XML versions and applies SQL to detect similarities. Experiments show that this approach is highly scalable and more efficient in terms of execution time, and it provides results comparable in quality to those yielded by previous approaches.

Keywords: XML Similarity, Change Detection, Keys, Join

1. INTRODUCTION

XML has become the universal standard for data representation and semi-structured data exchange due to its simplicity, platform independence, and ease of processing [7]. XML sources may have similar content, but this content may be described differently in each source using different tag names and structures. Such disparities are apparent in bibliographical data sources such as DBLP [19] and SIGMOD Record [1]. Not only is integrating similar XML documents from different data sources important to query systems and search engines, but it also gives users access to more complete and useful information. In an environment of frequently changing online information, the ability to quickly detect changes between two document versions is important for the maintenance of up-to-date integrated information.

Because XML documents not only encode structure but also store data, accurate measurement of similarities among them requires evaluations of similarities in both content and structure. A simple count of the number of common occurrences of XML elements or PCDATA between two XML documents is enough to identify similarity of structure and content, but this method can be very time-consuming. There exist several methods [9, 10, 11, 15, 16, 17] to address this problem. This paper also proposes an efficient method called XML-SIM [15] to measure the semantic content and structural similarity of XML documents. This method uses information theory to identify semantic similarity and subtree keys for comparison.

As XML documents change, a change detection mechanism can be used to perform an XML join. . This work developed an approach called XML-CHANGE, based on XRelChangeSQL [14], to detect changes between versions using SQL. This approach is much more efficient than comparison algorithms in main memory.

This paper develops a technique called XML-SIM-CHANGE by incorporating XML-CHANGE and XML-SIM to find similarities of structure and content among XML documents. The differences found in the change detection phase are used to reduce the number of nodes requiring comparison between two versions. The objective was to design, implement, and evaluate the technique that can detect similarity both content and structure in XML documents that have been changed. The method introduced here uses a

similarity matching algorithm and evaluates the changes detected between two versions. The contributions of this paper can be summarized as follows:

1. It proposes a framework called XML-SIM-CHANGE for detecting XML document similarity after documents have changed. It describes a method to detect changed subtrees and to match the changed subtrees using keys.
2. It also describes experiments to evaluate the new framework and compare it with XML-SIM using DBLP and SIGMOD Record, two bibliographical data sets.The results show that the new approach combining XML-SIM and XML-CHANGE can detect XML document similarity among versions much faster than XML-SIM alone and provide results of comparable quality.

2. RELATED WORK

XML documents are considered collections of items represented in XML tree form. In most XML document matching algorithms [9, 10, 11, 15, 16, 17], an XML document is fragmented into small independent items, or entities, representing real-world objects called subtrees. Similarities between the subtrees of two XML documents are measured to determine which subtree pairs are similar beyond a given threshold. These subtrees are considered matched pairs, which can be integrated into a single XML document.

XML documents can be similar in terms of structure alone or in terms of both content and structure. The structural similarity alone is used primarily in document clustering and in change detection. Most algorithms that identify structural similarity are based on tree-edit distance [5, 21]. Basically, such algorithms find the sequence of edit operations that can transform one tree into another at the lowest possible computational cost. However, tree-edit distance has not been used on a large scale due to its complexity and high computational cost. To integrate XML documents, similarities in both content and structure must be considered. Previous work [17] demonstrated that XML-SIM outperforms LAX [10] and SLAX [11]. The latter methods determine the degree of tree similarity based on the mean value of the degree of similarity between matched subtrees. The subtrees are clustered into subtrees based on the depth of the XML document and the

number of instances it contains. The subtrees are then used to calculate the similarity. Although LAX and SLAX outperform schemes based on edit distance, they ignore semantic information available such as keys and rely instead on the detection of subtrees or clustering points, which does not work for all types of XML data.

Three approaches permit identification of similarity between elements: string matching, edit distance, and semantic similarity. These approaches are effective and widely used for measuring similarity. String matching is simple to implement, but it fails to detect some similar strings. As mentioned above, the tree-edit distance is time-consuming and the results may not be semantically accurate [3, 21]. The longest common subsequence (LCS) approach is similar to the edit-distance method [2]. It finds the longest sequence of tokens common to two strings, but it may fail to identify connections between texts. Semantic similarity methods [8, 13] have been introduced capture the meaning of words. These methods are based on natural language processing (NLP) techniques that compute the degree of similarity between words (concepts). The similarity of two concepts is defined as the maximum of the information content of the concept that subsumes them in the taxonomic hierarchy.

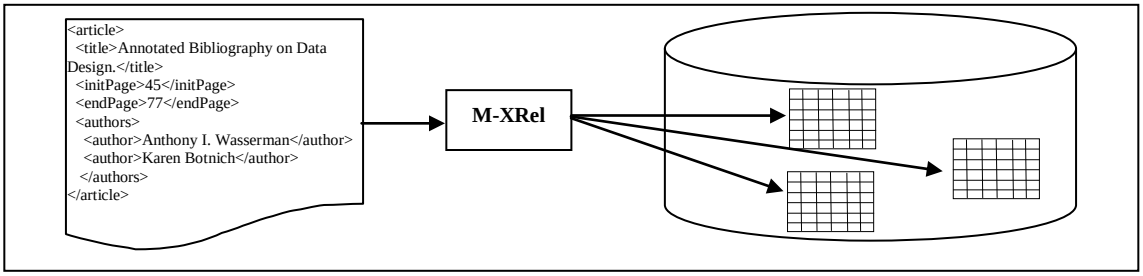
Keys are fundamental to data models and conceptual design. Along with semantic similarity, XML keys assist in the subtree matching [4]. Identification of keys can help in identify the real-world objects in XML documents, thus reducing dramatically the number of comparisons. Since most of the XML data is data-centric (i.e., derived from the relational data model), keys can best be used to improve evaluation of subtree similarity. Several previous studies [15, 16, 17] have describes the use of keys to find subtree matches.

3. M-XRel

To increase the scalability of XML document similarity and change identification, documents are stored in a relational schema using M-XRel, a modified version of XRel. XRel [20] is a method of storing and retrieving XML documents using relational databases. Most methods of detecting similarity in XML documents are focused on constructing document object model (DOM) trees. The tree comparison approach is not

efficient for handling large XML documents because the entire trees of both documents must reside in the main memory during the comparison process. XRel uses a model-mapping approach to decompose an XML document into nodes on the basis of its tree structure and store it in relational tables according to the node type with information on the path from the root to each node, as shown in Figure 1a.

The basic M-XRel schema consists of the five relational schemas shown in Figure 1b. They are similar to XRel schema but two additional columns, the 'parentid' and parent 'start' region, are added to the 'Element' and 'Text' tables. These extra fields improve the efficiency of the change detection process because comparisons of relational inequality can be replaced by equality comparisons in order to detect the parents of XML leaf nodes and nonleaf nodes.



(a)

Document(docID, document)
Element(docID, pathID, start, end, index, reindex, **parentid**, **pstart**)
Attribute(docID, pathID, start, end, value)
Text(docID, pathID, start, end, value, **parentid**, **pstart**)
Path(pathID, pathexp)

(b)

Figure 1. M-XRel storage: (a) storing XML documents (b) M-XRel schema

The database attributes are described in Table 1. An occurrence of an element or a leaf node is identified by its region and stored in the relations 'Element' and 'Text'. To identify each attribute node, the attribute name is stored as the suffix of the simple path expression of an attribute node, and the attribute value is stored in the relation 'Attribute'.

The database attribute “pathexp” in the relation Path stores simple path expressions. The ancestor-descendant relationships and the ordering of nodes can be found using ‘parentid,’ parent ‘start’ region, and ‘index’ value because these regions define the range of nodes (elements, leaf node, and attribute values) in the XML document.

Table 1. M-XRel field descriptions

Field descriptions	
docid	Document ID
Parentid	Parent’s path expression ID
pstart	Parent’s start value of the region
pathid	Path expression ID
pathexp	Path expressions of XML elements
start	Start value of the region
end	End value of the region
index	Forward index
reindex	Reverse index
value	Leaf node and attribute values

4. XML-SIM

Previous work [16] proposed XML-SIM for evaluating the similarity between XML documents. Two XML documents, Doc_b and Doc_t, are the base and target documents, respectively. They are stored in a relational database using M-XRel as described in Section 3. The following definitions apply to the XML document tree:

Definition 1. XML Document tree: *An XML document tree is a triple $T_i = (V, v_0, E)$, where V is the set of nodes, v_0 is the root node, and E is the set of edges in the tree T_i. Let T_b be a base document tree from the base document Doc_b, and T_t be a target document tree from the target document Doc_t.*

Definition 2. Path expression: *Any node v_i can be identified its location within a tree T_i by a path expression or path signature p_{v_i}. This path expression consists of one node or a series of nodes from the node set V separated by "/".* Path expressions are employed to measure semantic structural similarity.

The XML-SIM algorithm consists of three phases, illustrated in Figure 2: (1) subtree generation and validation, (2) key generation and subtree filtering and (3) similarity detection.

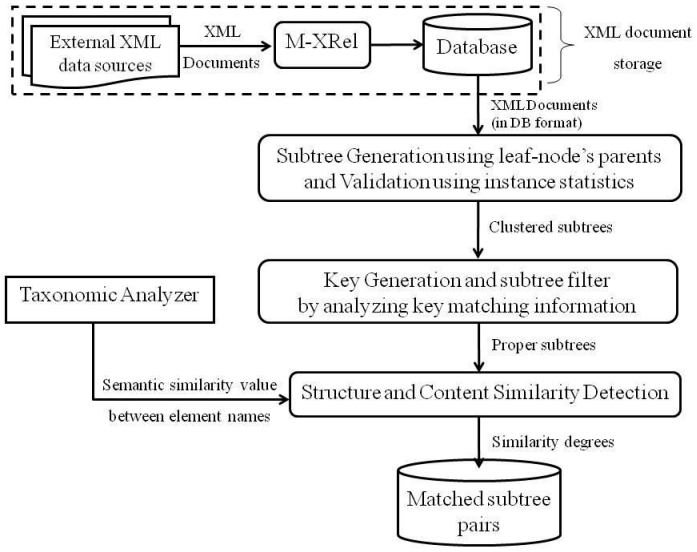


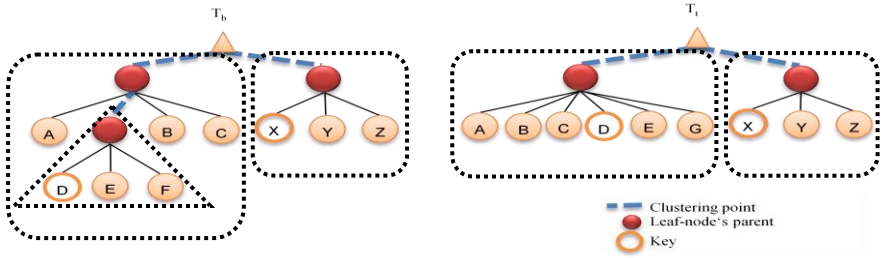
Figure 2. XML-SIM framework

4.1 Subtree generation and validation

The first phase is to extract subtrees from XML documents because an XML document is considered a collection of items. Since data in XML documents are stored at the leaf nodes, they are retrieved by using parents of leaf nodes to group related information together as a subtree. The leaf node parent is described in Definition 3.

Definition 3. Leaf-node Parent: *For a document tree T_i with a node set V and an edge set E, v_p is a leaf node parent, if $v_p \in V$ and $(v_l, v_p) \in E$, where v_p is the parent of the leaf node v_l.* Figure 3a shows leaf node parents in an XML document. The edges above the leaf node parents are considered clustering points; they will be removed to identify subtrees.

Extracted subtrees representing an independent object should contain nodes representing various types of information (e.g., author, title, and date), rather than a single type of information (e.g., an "authors" list). Based on subtree element structure [18], the concept of instance statistics is applied to determine whether the leaf-node parent element and the leaf-node elements of its children preserve a loose one-to-one relationship. The extracted subtrees are stored in the subtree relation. The clustered subtrees are categorized into one of two groups: simple or complex. A simple subtree has only one leaf-node parent; a complex subtree contains more than one leaf-node parent.



(a)

Subtree (docID, ppathID, pstart, pend, pathid, start, end, value**, key**, **subtreeid**)

(b)

Figure 3. Subtree: (a) clustering by leaf-node parents and (b) relation

4.2 Key generation and subtree filter

This phase defines subtree keys. A subtree key is modeled as an XML attribute, which is one of leaf nodes in a subtree. It has a unique value and is able to identify other attributes in its subtree.

Definition 4. Subtree key: *A subtree key is a leaf node* v_{key} *that has a unique value compared with any leaf nodes* v_l *having the same path expression* $p_{v_{key}}$*, where* $p_{v_{key}}$ *is the path expression of the node* v_{key}.

The SQL query shown in Figure 5 identifies subtree keys for XML documents by retrieving from the *text* relation unique values that can be used to distinguish among items.

```
SELECT docid, pathid, value
FROM text
GROUP BY docid, PathID,Value
HAVING Count(Value) = 1
```

Figure 4: SQL query to identify leaf nodes as keys

The labels associated with the leaf nodes returned by the query in Figure 4 are considered subtree keys. On the matched records (according to their docid, pathid, and value) in the *subtree* relation, "Y" is flagged as the attribute key.

In the next step detection of subtree similarity using the subtree keys overlaps with subtree filtering. The subtree keys previously identified are used to match subtrees by comparing the subtrees whose leaf nodes (labels) are marked as keys and have identical values. Although this comparison disregards the structure of the leaf nodes; the matching results can be analyzed to determine which subtree level is inappropriate for comparison. The matching information is analyzed by comparing the number of subtree matches with the median number of alternate keys. This process is based on the assumption that a complex subtree may contain many simple subtrees that in turn contain alternate keys. Such subtrees may cause several inappropriate matches, and they are considered inappropriate. To eliminate the inappropriate subtrees, threshold is calculated from the median number of alternate subtree keys. The subtrees causing a number of multiple matchings higher than this threshold value are eliminated.

4.3 Similarity detection

The subtree keys found in the previous step are compared to detect appropriate matched subtree pairs. This comparison considers both the structure and content of the base and target XML subtrees. Unmatched subtrees will be compared in a later phase.

Notation: *For any subtree $s_i = (V_i, v_p, E_i)$ rooted by a distinct label of node v_p, let $V_l = \{v_{l1}, v_{l2}, \dots, v_{ln}\}$ be a collection of leaf nodes in s_i if $V_l \in V_i$. If $P_l = \{p_{l1}, p_{l2}, \dots, p_{ln}\}$ represents a collection of path expressions (see in Definition 2) of the leaf nodes in V_l, which has n elements.*

To determine the semantic similarity of two paths, all p_{lj} in the base subtree where $1 \le j \le n_b$ are compared with all p_{lk} in the target subtree where $1 \le k \le n_t$ and the terms n_b and n_t represent the number of leaf nodes in the base subtree s_j and the target subtree s_k respectively. Measurement of the similarity between paths p_{lj} and p_{lk} requires comparison of the node labels of both.

Definition 5. Node Label Semantic Similarity Degree (NSSD). *For each pair of path expressions p_{lj} and p_{lk}, let $V_j = \{v_{j1}, v_{j2}, \dots, v_{jn_b}\}$ and $V_k = \{v_{k1}, v_{k2}, \dots, v_{kn_t}\}$ denote a series of nodes in p_{lj} and p_{lk} respectively. The node label semantic similarity degree is based on methods of Jiang and Resnik [7, 12] and defined as*

$$NSSD\left(l(v_j), l(v_k)\right) = 1 - \frac{IC\left(l(v_j)\right) + IC(l(v_k)) - 2sim(l(v_j), l(v_k))}{2}. \quad (1)$$

The IC value is calculated by considering the negative log of the probability:

$$IC(c) = -log\ p(c) \quad (2)$$

where $p(c)$ is the probability of having c in a given corpus and c is a concept in WordNet. The use of the negative likelihood is based on the notion that the more likely the appearance of a concept, the less information it coveys.

The function $sim(c_1, c_2)$ is evaluated by using the subsumer $S(c_1, c_2)$ of c_1, c_2:

$$sim(c_1, c_2) = max_{c \in S(c_1, c_2)} IC(c). \quad (3)$$

Definition 6. Path Semantic Similarity Degree (PSSD): *A path semantic similarity degree is expressed by the ratio of the sum the average NSSD for each node v_j in the path expression $p_{lj}$and the number of nodes in the path expression series. It can be expressed as:*

$$PSSD\left(p_{lj}, p_{lk}\right) = \frac{\sum_{j=1}^{n_b} avg(NSSD_j)}{n_b} \quad (4)$$

where $avg(NSSD_j)$ is computed from:

$$avg(NSSD_j) = \frac{\sum_{k=1}^{n_t}(NSSD(l(v_j), l(v_k))}{n_t}. \quad (5)$$

Definition 7. Matched Path Pair (MPP): A matched path pair is the pair with the highest PSSD value:

$$MPP(p_{lj}, p_{lk}) = max_{1\le j\le n_b; 1\le k\le n_t}\ (PSSD(p_{lj}, p_{lk})). \quad (6)$$

Definition 8. Selected Path Pair: *The selected path pair is the path expression with an MPP value greater that a given threshold* τ_p

At this point, all path expressions at the leaf-node levels have been evaluated and selected. The selected paths (see Definition 8) will be used to determine the similarity in subtree content.

Definition 9. Subtree Similarity based on structure and content: *The PCDATA value of each subtree* $t_{bi}(1 \le i \le k_b)$ *(content approach) is compared with those of the subtrees* $t_{tj}(1 \le j \le k_t)$ *based on the selected path (structure approach) to identify the proper matched subtree pair (MSP).*

Such a comparison based on content and structure can be done simply using loops, but this method is time consuming if there are many subtrees. Instead of loops, the approach introduced here uses an SQL query to retrieve subtree pairs, a much faster process. The subtree pairs thus identified, which are based on the same leaf-node parent, intersect to find the subtree pair that best satisfies the conditions, which have the same PC data content and a similar structure. The matched subtree pairs are stored in a relation called matching, the schema of which is shown in Figure 5.

Matching (base_docid, base_ppathid, base_subtreeid, target_docid, target_ppathid, target_subtreeid)

Figure 5: Matching relation

5. XML-SIM-CHANGE Framework

This work proposes a time-efficient technique to detect subtree similarity between two versions of the same XML document, not by running full pair-wise comparisons but by comparing the changes with the previous matching results. The framework of this XML-SIM-CHANGE approach is described here, followed by additional definitions, the details of each component, and the algorithm.

5.1 Overview of XML-SIM-CHANGE

This approach permit similarity detection using the results of change detection for two versions of an XML document and the semantic similarity described in Section 4.

Figure 6 shows the initial step in this approach, comparing two XML documents, Doc_bV_1 and Doc_tV_1, with some similar content from two heterogeneous data sources (referred as the base and target data sources). XML-SIM clusters the documents into subtrees using the leaf-node parents and compares the clustered subtrees using subtree keys and degrees of semantic similarity. This process yields matched subtree pairs,which can be joined in order to integrate the XML documents.

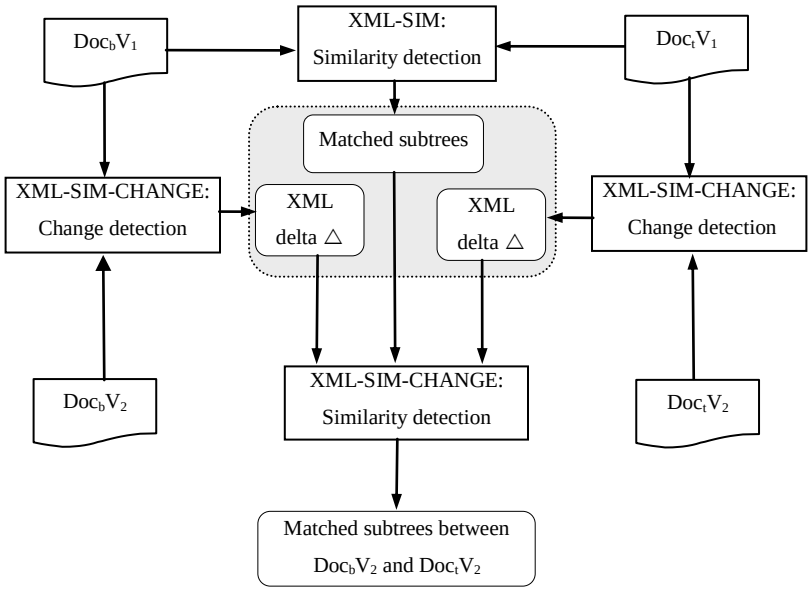


Figure 6: Overview of XML-SIM-CHANGE

When one or both XML documents have been altered (from Doc_bV_1 to Doc_bV_2 or from Doc_tV_1 to Doc_tV_2) after integration, the changes are detected by identifying deleted and inserted nonleaf nodes and leaf-node changes (i.e., delete, insert, and update). These changes, along with the results of XML-SIM, are used to compute the similarity between the two versions of the XML documents.

5.2 Finding XML document changes

Two documents Doc_iV_1 and Doc_iV_2 from the same data source Doc_i are stored in a relational database using M-XRel (as discussed in Section 3), and changes have been detected in each. To compare the two, this approach follows the three steps shown in Figure 7: (1) finding matching subtrees, (2) detecting deleted and inserted nonleaf nodes, and (3) detecting leaf node changes. The results will reveal the changes.

5.2.1 Finding Matching subtrees

First, subtrees s_j from the old version of an XML document and s_k from the new version are compared, where (i) $1 \le j \le n_{V1}$ and (ii) $1 \le k \le n_{V2}$ by matching leaf-node parents among subtree keys v_k (unique leaf-node value) in Doc_iV_1 and Doc_iV_2.

Definition 10. Subtree pairs matched by subtree keys (SMK): *Each subtree* $s_j (1 \le j \le n_{V1})$ *in* Doc_iV_1 *is compared with the subtrees* $s_k (1 \le k \le n_{V2})$ *in* Doc_iV_2 *using their subtree key (see Definition 2). The subtrees are matched if:*

(i) their leaf-node parents have the same path expression, i.e., $pathid\left(v_{p_{s_j}}\right) = pathid\left(v_{p_{s_k}}\right)$ *and*

(ii) there exist leaf nodes designated as subtree keys and having values common in both the versions such that $pathid\left(v_{key_{s_j}}\right) = pathid\left(v_{key_{s_k}}\right)$ *and* $value\left(v_{key_{s_j}}\right) = value\left(v_{key_{s_k}}\right)$.

Definition 10 identifies the best match between leaf-node parents and avoids unnecessary comparisons between duplicates later. The unmatched subtrees are compared using degrees of subtree similarity as discussed below.

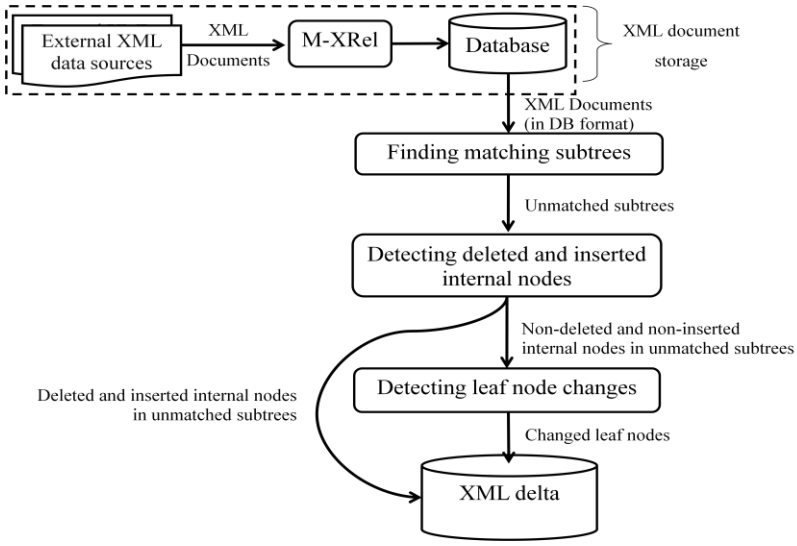


Figure 7: Framework for identifying changes between two versions

Definition 11. Subtree Similarity Degree (SSD): *Assume n is the number of leaf nodes having the same PCDATA value and the same pathid. Let n_j represent the number of leaf nodes in subtree s_j, and let n_k represent the number of leaf nodes in subtree s_k:*

$$SSD(s_j, s_k) = \frac{2 \times n}{n_j + n_k}. \quad (7)$$

SSDs having values higher than a defined threshold are stored in the relational database. These values will be used in the next step.

Definition 12. Subtree pairs Matched by SSD (SMS): *The matched subtree pair s_j and s_k is the pair that has the maximum degree of subtree similarity as defined above (Definitions 11):*

$$SMS[i] = \max\left(SSD_i(s_j, s_k)\right). \quad (8)$$

The subtree matches including SMK and SMS are then distinguishable from unmatched subtrees.

5.2.2 Detecting deleted and inserted nonleaf nodes

The deleted and inserted subtrees is sets of matched subtrees are detected as follows:

Definition 13. Deleted and inserted nonleaf nodes: *Let* S_1 *and* S_2 *be the sets of all subtrees from the XML document version* Doc_iV_1 *and the XML document version* Doc_iV_2 *respectively. Assume* MS_{v1} and MS_{v2} are the sets of matched subtrees $\{SMK \cup SMS\}$ *from* Doc_iV_1 *and* Doc_iV_2. The set $D_{inter} = \{S_{v1} - MS_{v1}\}$ is then the set of deleted nonleaf nodes, and the set $I_{inter} = \{MS_{v2} - S_{v2}\}$ is the set of inserted nonleaf nodes.

Since all leaf-node parents were matched in the previous step, the unmatched leaf-node parents in the first version can be identified deleted nonleaf nodes, and the unmatched leaf-node parents in the second version must be inserted nonleaf nodes.

5.2.3 Detecting leaf-node changes

The deleted and inserted leaf nodes are those whose parents have been identified as matched subtrees but whose values are not matched with any leaf-node values in the matched subtree. The deleted and inserted leaf nodes are identified as follows:

Definition 14. Deleted and inserted leaf nodes: *Let* ML_1 *and* ML_2 *be the sets of all exactly matched leaf nodes. Let* L_{v1} *and* L_{v2} *denote the sets of all matched leaf nodes from the subtree matching step. The deleted leaf nodes can be identified by the set* $D_{ln} = \{L_{v1} - ML_{v1}\}$, *and the inserted nodes can be detected by the set* $I_{ln} = \{L_{v2} - ML_{v2}\}$.

The results of Definition 14 provide the updated leaf nodes. Among inserted and deleted leaf nodes, those with the same signature and the same matching parent are considered updated.

Definition 15. Updated leaf nodes: *Let* U_{2ln} *denote the set of updated nodes, which can be found if there exist leaf nodes* $v_j \in D_{1ln}$ *and* $v_k \in I_{2ln}$, *such that (i)* $pathid(v_j) = pathid(v_k)$, *(ii)* $pathid\left(parent(v_j)\right) = pathid(parent(v_k))$, *and (iii)* $value(v_j) \neq value(v_k)$.

This step reveals the change that is set $\{D_{ln} \cup I_{ln} \cup U_{ln} \cup D_{inter} \cup I_{inter}\}$. This change will be used in the next step.

5.3 Detecting subtree similarity between new versions of XML documents

The base XML document Doc_bV_1 and target XML document Doc_tV_1 are analyzed using XML-SIM to find the best-matched subtree pairs (MSP). When the XML documents Doc_bV_1 and Doc_tV_1 are modified to Doc_bV_2 and Doc_tV_2 respectively, the old and new versions are detected in order to identify the changes in each document.

The XML-SIM-CHANGE approach uses the matching results from XML-SIM and the results of change detection to find the best-matched pair in the new version of the XML document. This procedure is split into two steps: preprocessing and comparing the matched subtree pairs with the changes.

5.3.1 Preprocessing

Once the subtrees obtained by XML-SIM are filtered, the new versions of both documents are clustered into subtrees using the selected leaf-node parents.The changes identified by XML-CHANGE are then mapped to the clustered subtrees. The mapping can simply use the region elements (the start and end attributes) defined in M-XRel.

The subtrees matched with the changes are marked as ‘updated leaf node,’ ‘deleted leaf node,’ inserted leaf node,’ ‘deleted internal node,’ or ‘inserted internal node’. These subtree flags identify the subtree level changed.

5.3.2 Comparing the matching with the change

This section explains how to find similarity in the new versions of XML documents. Figure 8 shows the algorithm of XML-SIM-CHANGE. First, a match is found between the subtrees marked ‘update’ and those from previous matching results by comparing the path signature of the root of subtrees, the path signature of the node identified as a subtree key, and the value of the subtree key. If a match is found, the subtree from the matching results is updated by its change type. Next, the deleted subtrees are addressed; they have either a deleted leaf nodes or deleted internal nodes. If a match is found between deleted subtrees and the set of matched subtrees, the next step is to determine whether the deleted node is the root of the subtree or a nonroot node. If it is the root, the subtree is removed from the matching set. If not, only the deleted node is removed.

Algorithm XML-SIM-CHANGE

Input: $M_{old}\{(s_j, s_k)\}$ //Matching result from measuring the similarity of Doc_bV1 and Doc_tV1
$\Delta = \{D_{1ln} \cup I_{2ln} \cup U_{2ln} \cup D_{1inter} \cup I_{2inter}\}$ //the change
Output: $M_{new}\{(S_{j,new}, S_{k,new})\}$ //Matching result from measuring the similarity of Doc_bV2 and Doc_tV2

//***Pre-processing***
//Clustering an XML document by the selected leaf-node parent in XML-SIM
$S_{b,v2}$ = Cluster(Doc_bV2); //$S_{b,v2}$ is the set of subtrees in Doc_bV2
$S_{t,v2}$ = Cluster(Doc_tV2); //$S_{t,v2}$ is the set of subtrees in Doc_tV2
//Matching the subtrees with the change
S_Δ = changeMatching($S_{b,v2}$, $S_{t,v2}$, Δ) //$S_\Delta$is the set of subtrees having some change
//***Detecting the matching for the documents Doc_bV2 and Doc_tV2***
for each s_i having the flag as 'update' in S_Δ
{ if (matchSubtree(s_i, the set of all subtrees in M_{old})) //check for a subtree match
update s_i to the matched subtree in M_{new};
}
}
for each s_i having the flag as 'delete' in S_Δ
{if (matchSubtree(s_i, the set of all subtrees in M_{old}))
{if (flag == 'deleted internal node' && deleted node == the root of s_i)
{ delete a pair having $s_j$as a match from M_{new};}
else{ Remove the deleted node from subtree in M_{new}; }
}
}
for each s_i in Doc_bV2 having the flag as 'insert' in S_Δ
{ if (matchSubtree($s_i, S_{j,v1}$))
{ insert a pair of ($s_i, s_{j,v1}$) into the set of M_{new} };
}
for each s_i in Doc_tV2 having the flag as 'insert' in S_Δ
{if (matchSubtree($s_i, S_{k,v1}$))
{ insert a pair of ($s_i, s_{k,v1}$) into the set of M_{new} };
}
Module: matchSubtree(subtree s_i, setOfSubtree S_j){
rs = select count(*) from S_j
where $pathid(v_{p,si}) = pathid(v_{p,sj})$ and $pathid(v_{key,si}) = pathid(v_{key,sj})$
and $value(v_{key,si}) = value((v_{key,sj})$
if (rs == 1){ //found a match
return true;
}
return false;
}

Figure 8: XML-SIM-CHANGE algorithm

For the inserted nodes, the document must be compared with unmatched subtrees in previous version. If a match is found, the matched subtree pair from the previous version and the inserted subtree is added to the matching set. After processing the algorithm, the matched subtree pairs are up to date with the new versions.

6. XML-SIM-CHANGE Performance Evaluation

This work evaluated the efficiency and effectiveness of XML-SIM-CHANGE algorithm compared with the pure XML-SIM algorithm. (Previous work [15] has shown it outperforms other comparable approaches.)

6.1 Experimental setup and data sets

Experiments were conducted using an Intel Core 2 Duo 2.20GHz CPU processor with 4GB of RAM running on Windows XP Professional with Sun JDK 1.6.0_02 and an Oracle Database 10g Standard Edition. M-XRel was used to store the XML documents on the Oracle 10g. Implementation was tested using the real data sets Sigmod Record and DBLP, which were modified to create new versions of the documents with various degrees of changes. The data sets were divided into three groups: large, medium, and small. Each group included documents with three different levels of change: 25%, 50%, and 75%. For simplicity, the changes in leaf nodes or internodes were divided into two groups: deletion and insertion, because update operations here were considered a combination of a deletion and an insertion.

This experiment used XML documents from SIGMOD Record [1] (referred to as doc1) and DBLP [19] (referred to as doc2). The term doc1.V1 represents the XML document from doc1 before any changes, and doc1.V2 represents the XML document from doc1 after changes. Tables 2 and Table 3 describe the data sets. Each XML document contains a collection of items or subtrees; in this case, the items from doc1 represent <article> in SIGMOD Record and those in doc2 represent <inproceedings>, <incollection>, <book>, <article>, <www>, <masterthesis>, <phdthesis>, or <proceedings> in DBLP. Doc2.V2 is represented by its size and the percentage of changes for each data set.

6.2 Experimental results

Previous work has compared XML-SIM to existing approaches [15, 16, 17], demonstrating that it outperforms XDI-CSSK and XDoI, both of which perform better than LAX and SLAX [10, 11]. This section presents the results of the experiments described here including execution time and accuracy of the similarity detection. First, the speed of XML-SIM-CHANGE was evaluated by comparing it with pure XML-SIM. The similarity threshold τ was 0.7 in both approaches.

Table 2. Controlled data sets

Document	Small document		Medium document		Large document	
	File size (KB)	# of subtrees	File size(KB)	# of subtrees	File size (KB)	# of subtrees
Doc1.V1	7	20	482	1504	482	1504
Doc1.V2*	7	20	482	1504	482	1504
Doc2.V1	12	11	679	1337	10 MB	31016

*Note: the Doc1.V1 and Doc1.V2 are the same because there was no change in the first document (Doc1).

Table 3. Data set descriptions for Doc2.V2

Group/Change	Data set (size-% change)	File size (KB)	# of subtrees
Large/Delete	L25D	8 MB	25997
	L50D	5 MB	9199
	L75D	3 MB	8997
Large/Insert	L25I	12MB	32944
	L50I	15MB	41074
	L75I	18MB	46092
Medium/Delete and Insert	M25	683	774
	M50	678	768
	M75	669	772
Medium/Delete only	M25D	487	564
	M50D	299	350
	M75D	114	131
Medium/Insert only	M25I	812	919
	M50I	1006	1154
	M75I	1166	1338
Small/Delete and Insert	S25	14	11
	S50	13	11
	S75	12	11

Figure 9 shows how well XML-SIM-CHANGE detected similarity in content and semantic structure in the new version of Sigmod Record and DBLP. Figure 9(a) shows the execution time. Here, the pure XML-SIM approach performed better than XML-SIM-CHANGE for changes in small documents because the overhead involved in detecting

the changed nodes is higher for small documents in which few subtrees require comparison. However, XML-SIM-CHANGE dramatically outperformed XML-SIM for larger XML documents. Figure 9(a) shows that for a small document with 25% change, a file size 14KB (making it largest file among the small documents) significantly affected the execution time for both the approaches. As indicated in Figure 9(c) and (e), for medium and large documents, if fewer than 50% of nodes had been changed then the execution time for XML-SIM-CHANG was much better than for pure XML-SIM approach. Figure 9(b) shows the execution time for medium-size documents with both insertion and deletion. Since the document sizes (Medium-25%, Medium-50%, and Medium-75%) were almost the same as those shown in Table 2, the execution times for each pair in XML-SIM did not vary. Figure 9(c) and (e) show when nodes in the old document version have been deleted, the size of the new version becomes smaller, which decreases the execution time of XML-SIM. Similarly, Figure 9(d) and (f) show that when nodes have been added, the size of the new version grows, which prolongs the execution of XML-SIM. However, XML-SIM-CHANGE performs better because it benefits from the change results and thus avoids unnecessary comparisons of all pairs in both XML documents. The change detection process is quick compared to the matching process in XML-SIM because it uses regions stored by M-XRel in the DBMS. Thus, the new approach is much more scalable and thus able to handle very large documents with both insertion and deletion of nodes.

The quality of the matching results was also evaluated. Subtree matching was evaluated as Sn/An, where *Sn* is the number of subtrees matched by a given approach and *An* is the number of actual matched subtrees. Figure 10 shows the quality of results for both approaches. The results of both XML-SIM and XML-SIM-CHANGE are the same because both use the same matching method. They are able to identify the matched path pair and subtree keys for both the XML documents by taking advantage of subtree keys and filters. The quality of results for XML-SIM-CHANGE is based on the matching subtree in the change detection phase, which relies mainly on leaf-node value matches.

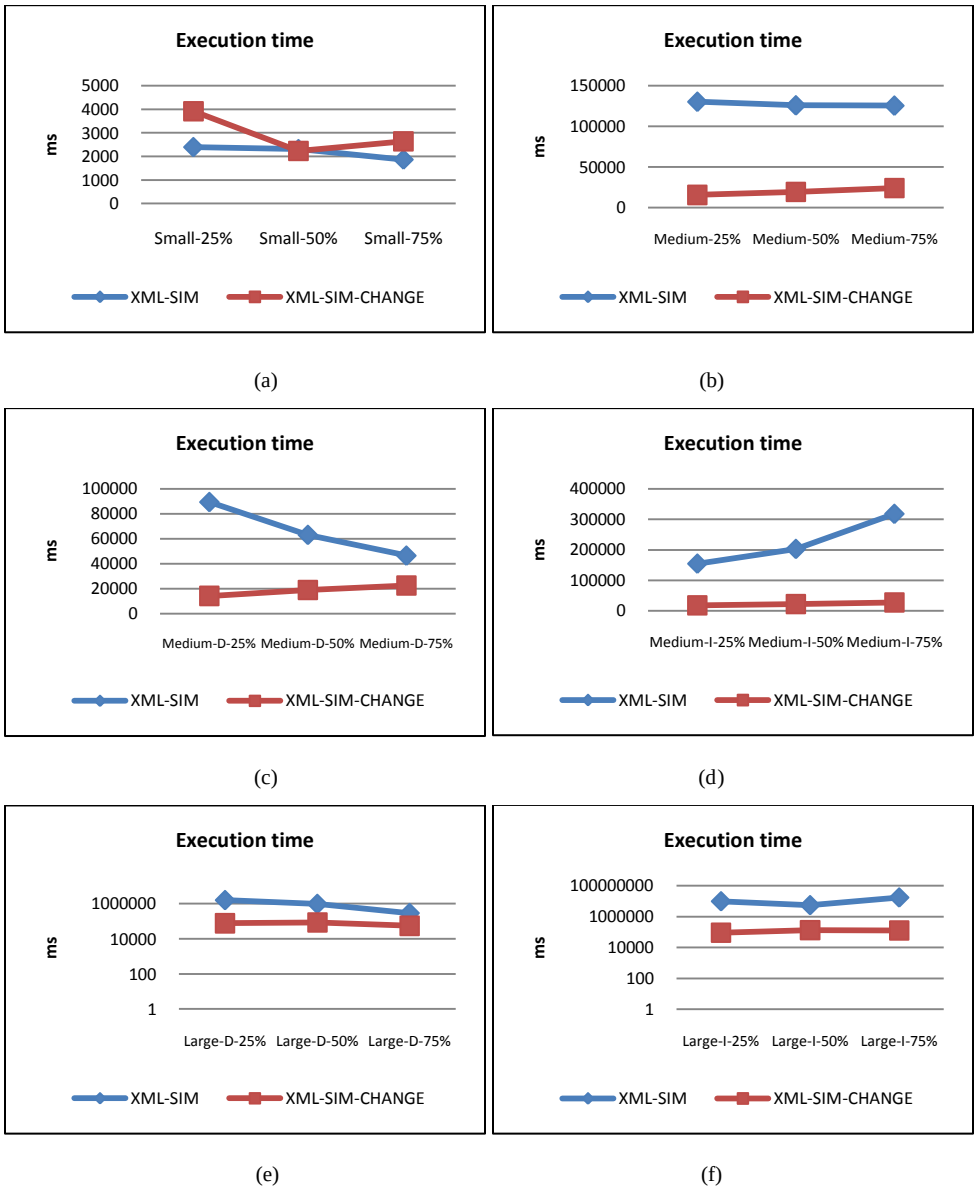


Figure 9: Execution time of: (a) small data sets with the change of insertions and deletions (b) medium data sets with the change of insertions and deletions (c) medium datasets with the change of deletions (d) medium data sets with the change of insertions (e) large datasets with the change of deletions (f) large data sets with the change of insertions

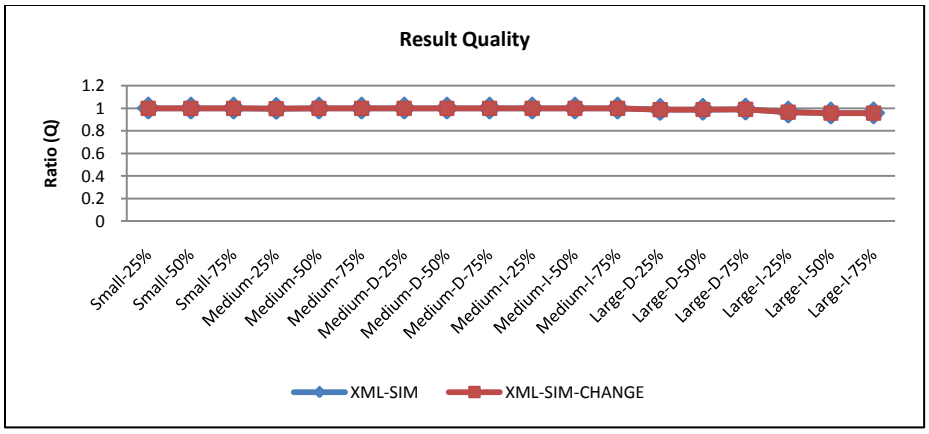


Figure 10: Result quality

7. CONCLUSIONS AND FUTURE WORK

This work has proposed a technique called XML-SIM-CHANGE for finding XML document similarity after such documents have been changed. The technique relies on collaboration between the change detection method and subtree matching. It avoids unnecessary comparisons by taking advantage of subtree keys and filters in XML-SIM and thus comparing only subtrees with changes in order to find the best matched subtree pairs in the new versions. Future work will consider different types of XML documents (e.g., shallow, semi-shallow, deep) and apply this technique to measure the impact of these types of XML schema.

8. REFERENCES

[1] *ACM* SIGMOD *Record in XML. (n.d.).*, http://www.acm.org/sigmod/record/xml

[2] Apostolico, A., & Galil, Z.: *Pattern Matching Algorithms.* Oxford University Press, USA. (1997)

[3] Bille, P.: *Tree edit distance, alignment distance and inclusion.* Citeseer: IT Univ. of Copenhagen. (2003)

[4] Buneman, P., Davidson, S., Fan, W., Hara, C., & Tan, W.: Keys for XML. *Computer Networks* , 473-487. (2002)

[5] Chawathe, S.: Comparing hierarchical data in external memory, pp. 90-101. Citeseer. (1999)

[6] Christiane, F.: *WordNet: An Electronic Lexical Database.* MA: MIT press Cambridge. (1998)

[7] *Extensible Markup Language (XML)*. In: World Wide Web Consortium (W3C): http://www.w3.org/XML/

[8] Jiang, J., Conrath, D.: Semantic similarity based on corpus statistics and lexical taxonomy. *Jiang, J.J. and Conrath, D.W.* , 19-33. (1997)

[9] Liang, W., Yokota, H.: A path-sequence based discrimination for subtree matching in approximate XML joins. *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, pp. 23-28. (2006)

[10] Liang, W., Yokota, H.: LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration. *Proceedings Of BNCOD 2005*, pp. 82-97. (2005)

[11] Liang, W., Yokota, H.: SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes. *IPSJ Digital Courier , 2*, 382-392. (2006)

[12] Pirro, G., Seco, N.: Design, Implementation and Evaluation of a New Semantic Similarity Metric Combining Features and Intrinsic Information Content. *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems*, pp. 1271-1288. Springer. (2008)

[13] Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. *Proceedings of the 14th Internaltional Joint Conference on Artificial Intelligence* , 448-453. (1995)

[14] Sathyanarayanan, S., Sanjay, M.: *XREL_CHANGE_SQL:A Change Detection System for Unordered XML documents.* Rolla, MO: University of Missiouri-Rolla. (2005)

[15] Viyanon, W., & Madria, S.: A System for Detecting XML Similarity in Content and Structure Using Relational Database. *18thACM International Conference on Information and Knowledge Management (ACM CIKM 2009),* 1197-1206. (2009)

[16] Viyanon, W., & Madria, S.: XML-SIM: Structure and Content Semantic Similarity Detection using Keys. *Proceeding of the 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2009),* 1183-1200. (2009)

[17] Viyanon, W., Madria, S. K., & S, B. S.: XML Data Integration Based on Content and Structure Similarity Using Keys . *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems* , 484-493. (2008)

[18] Weis, M.: Fuzzy Duplicate Detection on XML Data. *Proceedings of VLDB 2005 PhD Workshop,* (p. 11). (2005)

[19] *XML Version of DBLP*. (n.d.). Retrieved May 2006, from http://dblp.uni-trier.de/xml/

[20] Yoshikawa, M., Amagasa, T., Shimura, T., & Uemura, S.: XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology* , *1* (1), 110-141. (2001)

[21] Zhang, K., & D, S.: Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing* , 1245. (1989)

3. CONCLUSION

This dissertation presents a series of algorithms for detection of similarity in structure and content between XML documents. These algorithms generate more complete information by integrating similar documents.

The first algorithm, XDoI, offers a data-centric approach to clustering XML documents into subtrees using leaf-node parents. It introduces subtree keys to reduce dramatically the number of subtrees to be matched, thus improving the degree of similarity by reducing false positives. To increase scalability and circumvent the difficulty of loading very large XML trees into the main memory, XML documents are stored in a relational database using XRel.

The second algorithm is XDI-CSSK, which introduces filtering methods to prune the unnecessary clustered subtrees that are primarily responsible for the high computation costs of similarity measurement in XDoI. The algorithm uses leaf-node parents as clustering points and validates them using the concept of instance statistics, along with a taxonomic analyzer. The information thus required is used to purge subtrees in one document that are unrelated to those in the other document. The execution time for this approach is better than that of XDoI because semantic similarity plays a crucial role in precise computational similarity measures.

XML-SIM is an improvement over XDoI and XDI-CSSK. It focuses on the semantics associated with the child nodes in a subtree, thus reducing the number of subtree comparisons to be made. The main improvement of this approach is that it determines content similarity based on structural similarity, which in turn is determined using semantics. The execution time for XML-SIM makes it a dramatic improvement over XDI-CSSK and XDoI.

XML-SIM-CHANGE is addresses the challenges posed by changes in the content of XML documents. It combines a change detection mechanism with the results of subtree matching from XML-SIM to avoid unnecessary comparisons of subtrees within different XML versions of the same document. The results of XML-SIM can be categorized into two groups, matched and unmatched subtrees. It compares the changes that result from updates and deletions with the matched subtree group. Changes resulting

from additions are compared with subtrees from the unmatched subtree group. The experimental results show that this approach is faster and yields results comparable in quality to those of XDoI and XDI-CSSK.

Future research will seek to exploit semantic similarity to compare not only the structure of XML documents (i.e., elements, attributes, and labels), but also their content (i.e., the values of elements and attributes). Further, they will consider various types of XML documents (e.g., shallow, semi-shallow, and deep) in multiple domains and use this new technique to measure the impact of various types of XML schema.

BIBLIOGRAPHY

[1] Abiteboul, S. (2001, March 27). *Xyleme, a Dynamic Warehouse for XML Data of the Web*. Retrieved March 2010, from Institut National De Recherche En Informatique Et En Automatique: http://www.inria.fr/MULTIMEDIA/Didactheque/4-Docmnt-Didact/0004/XYLEMEP.HTM

[2] *ACM SIGMOD Record in XML*. (n.d.). Retrieved March 2006, from http://www.acm.org/sigmod/record/xml

[3] Aguilera, V., Cluet, S., Veltri, P., Vodislav, D., & Wattez, F. (2000). Querying XML documents in Xyleme. *Proceedings of the ACM-SIGIR 2000 Workshop on XML and Information Retrieval.* Athens, Greece.

[4] Altinel, M., & Franklin, M. (2000). Efficient filtering of XML documents for selective dissemination of information. *Proceedings of the 26th International Conference on Very Large Data Bases*, (pp. 53-64).

[5] Apostolico, A., & Galil, Z. (1997). *Pattern matching algorithms.* Oxford University Press, USA.

[6] Augsten, N., Bohlen, M., & J, G. (2005). Approximate matching of hierarchical data using pq-grams. *Proceedings of the 31st international conference on Very large data bases* (pp. 301-312). VLDB Endowment.

[7] Baeza-Yates, R., Navarro, G., Vegas, J., & De La Fuente, P. (2002). A model and a visual query language for structured text. *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings* (pp. 7-13). IEEE.

[8] Bertino, E., Guerrini, G., & Mesiti, M. (2004). A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications* 1. *Information Systems* , 23-46.

[9] Bille, P. (2003). Tree edit distance, alignment distance and inclusion. *IT Univ. of Copenhagen TR-2003-23* .

[10] Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., Siméon, J., et al. (2002). XQuery 1.0: An XML Query Language. *W3C working draft* .

[11] Bouchachia, A., & Marcus, H. (2007). Classification of XML Documents. *IEEE Symposium on Computational Intelligence and Data Mining, 2007. CIDM 2007*, (pp. 390-396).

[12] Bourret, R. (2005, September). *XML and Databases*. Retrieved March 2010, from Ronald Bourret: http://www.rpbourret.com/xml/XMLAndDatabases.htm

[13] Buneman, P., Davidson, S., Fan, W., Hara, C., & Tan, W. (2002). Keys for XML. *Computer Networks* , *39* (5), 473-487.

[14] Buttler, D. (2004). *A short survey of document structure similarity algorithms.* Citeseer.

[15] Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., & Tanca, L. (1999). XML-GL: a graphical language for querying and restructuring XML documents. *Computer Networks-the International Journal of Computer and Telecommunications Networking* , 1171-1188.

[16] Chawathe, S. (1999). Comparing hierarchical data in external memory. *Proceedings of the International Conference on Very Large Data Bases* (pp. 90-101). Citeseer.

[17] Christiane, F. (1998). *WordNet: An Electronic Lexical Database.* MA: MIT press Cambridge.

[18] Cobena, G., Abiteboul, S., Marian, A., & INRIA, R. (2002). Detecting changes in XML documents. *Proceedings. 18th International Conference on Data Engineering, 2002.*, (pp. 41-52).

[19] *CVS: Concurrent Version System*. (n.d.). Retrieved March 2010, from CVS: http://www.cvshome.org/eng/

[20] Dalamagas, T., Cheng, T., Winkel, K., & Sellis, T. (2006). A methodology for clustering XML documents by structure. *Information Systems* , 187-228.

[21] Daly, P. (2003, December 8). *XML Basics and Benefits.* Retrieved August 4, 2010, from Intranet Jounal: Building the Corporate Enterprise: http://www.intranetjournal.com/articles/200312/ij_12_08_03a.html

[22] de Keijzer, A. (2006). Probabilistic XML in Information Integration. *Proceedings of the VLDB2006 Ph. D. Workshop.* Seoul, Rep of Korea: Citeseer.

[23] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., & Suciu, D. (1998). *Xml-ql: A query language for xml*. Retrieved March 2010, from http://www.w3.org/TR/NOTE-xml-ql/

[24] Dorneles, C., Heuser, C., Lima, A., da Silva, A., & de Moura, E. (2004). Measuring similarity between collection of values. *Proceedings of the 6th annual ACM international workshop on Web information and data management* (pp. 56-63). ACM.

[25] *Extensible Markup Language (XML)*. (n.d.). Retrieved March 2010, from World Wide Web Consortium (W3C): http://www.w3.org/xml

[26] Fellbaum, C., & others. (1998). *WordNet: An electronic lexical database*. MIT press Cambridge, MA.

[27] Fuhr, N., & Großjohann, K. (2001). XIRQL: A query language for information retrieval in XML documents. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 172-180). ACM.

[28] Ghosh, S., & Mitra, P. (2008). Combining Content and Structure Similarity for XML Document Classification using Composite SVM Kernels., (pp. 1-4).

[29] Gong, X., Qian, W., Yan, Y., & Zhou, A. (2005). Bloom filter-based XML packets filtering for millions of path queries. *21st International Conference on Data Engineering, 2005. ICDE 2005. Proceedings* (pp. 890-901). IEEE.

[30] Guo, L., Shao, F., Botev, C., & Shanmugasundaram, J. (2003). XRANK: Ranked keyword search over XML documents. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (p. 27). ACM.

[31] Jaccard, P. (1901). Etude comparative de la distribution florale dans une portion des Alpes et des Jura.[A study comparing the distribution of flora in a portion of the Jura Alps] Bulletin. *Societe Vaudoise des Sciences Natirelles* , 547-579.

[32] Jiang, J., & Conrath, D. (1997). Semantic similarity based on corpus statistics and lexical ontology. *Proc. of Int. Conf. Research on Comp. Linguistics X, Taiwan.*

[33] Kade, A., & Heuser, C. (2008). Matching XML documents in highly dynamic applications. *Proceeding of the eighth ACM symposium on Document engineering* (pp. 191-198). Brazil: ACM.

[34] Kim, J., Peng, Y., Kulvatunyou, S., Ivezic, N., & Jones, A. (2008). A layered approach to semantic similarity analysis of XML schemas. *IEEE International Conference on Information Reuse and Integration, 2008. IRI 2008* (pp. 274-279). IEEE.

[35] Kriegel, H., & Schönauer, S. (2003). Similarity search in structured data. *Data Warehousing and Knowledge Discovery* , 309-319.

[36] Leacock, C., & Chodorow, M. (1998). Combining local context and WordNet similarity for word sense identification. *WordNet: An electronic lexical database* , 265-283.

[37] Leitão, L., Calado, P., & Weis, M. (2007). Structure-based inference of xml similarity for fuzzy duplicate detection. *Structure-based inference of xml similarity for fuzzy duplicate detection* (pp. 293-302). ACM.

[38] Li, Y., Bandar, Z., & McLean, D. (2003). An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on knowledge and data engineering , 15* (4), 871-882.

[39] Lian, W., Cheung, D. W.-l., Mamoulis, N., & Yiu, S.-M. (2004). An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *IEEE Transactions on Knowledge and Data Engineering , 16* (1), 82-96.

[40] Liang, W., & Yokota, H. (2006). A path-sequence based discrimination for subtree matching in approximate XML joins. *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, (pp. 23-28).

[41] Liang, W., & Yokota, H. (2005). LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration. *Proceedings Of BNCOD 2005*, (pp. 82-97).

[42] Liang, W., & Yokota, H. (2006). SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes. *IPSJ Digital Courier , 2*, 382-392.

[43] Lin, D. (1998). An information-theoretic definition of similarity. *Proceedings of the 15th International Conference on Machine Learning* (pp. 296-304). Citeseer.

[44] Maier, D. (1978). The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)* , 322-336.

[45] Malik, S., Trotman, A., Lalmas, M., & Fuhr, N. (2007). Overview of INEX 2006. *Comparative Evaluation of XML Information Retrieval Systems* , 1-11.

[46] Manning, C., Raghavan, P., & Schütze, H. (2008). *An introduction to information retrieval*. Cambridge University Press.

[47] Marian, A., Abiteboul, S., Cobena, G., & Mignet, L. (2001). Change-centric management of versions in an XML warehouse. *Proceedings of the international conference on Very Large Data Bases* (pp. 581-590). Citeseer.

[48] Nierman, A., & Jagadish, H. (2002). Evaluating structural similarity in XML documents. *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)* (pp. 61-66). Citeseer.

[49] Pirro, G., & Seco, N. (2008). Design, Implementation and Evaluation of a New Semantic Similarity Metric Combining Features and Intrinsic Information Content. *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems* (pp. 1271-1288). Springer.

[50] Rada, R., Mili, H., Bicknell, E., & Blettner, M. (1989). Development and application of a metric on semantic nets. *IEEE transactions on systems, man and cybernetics , 19* (1), 17-30.

[51] Rafiei, D., & Mendelzon, A. (1998). Fourier transform based techniques in efficient retrieval of similar time sequences. *PhD thesis, University of Toronto* .

[52] Rafiei, D., Moise, D., & Sun, D. (2006). Finding syntactic similarities between xml documents. *Proceedings of the 17th International Conference on Database and Expert Systems Applications* (pp. 512-516). Citeseer.

[53] Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *Proceedings of the 14th Internaltional Joint Conference on Artificial Intelligence* , 448-453.

[54] Robie, J., Lapp, J., & Schach, D. (1999, August). *XQL (XML Query Language)*. Retrieved April 2010, from ibiblio: the public's library and digital archive: http://www.ibiblio.org/xql/xql-proposal.html

[55] Salton, G. a. (1975). A vector space model for automatic indexing. *Communications of the ACM , 18*, 620.

[56] Salton, G., & McGill, M. (1983). *Introduction to modern information retrieval*. McGraw-Hill New York.

[57] Schlieder, T., & Meuss, H. (2002). Querying and ranking XML documents. *Journal of the American Society for Information Science and Technology* , 489-503.

[58] Shasha, D. a. (1995). Approximate tree pattern matching. *Pattern Matching in Strings, Trees and Arrays* , 341-371.

[59] Tai, K. (1979). The tree-to-tree correction problem. *Journal of the ACM (JACM)* , 433.

[60] Tekli, J., Chbeir, R., & Yetongnon, K. (2009). An overview on XML similarity: Background, current trends and future directions. *Computer Science Review , 3* (3), 151-173.

[61] Tekli, J., Chbeir, R., & Yetongnon, K. (2007). Efficient XML Structural Similarity Detection using Sub-tree Commonalities. *SBBD and SIGMOD DiSC* .

[62] Tekli, J., Chbeir, R., & Yetongnon, K. (2007). Structural similarity evaluation between XML documents and DTDs. *LECTURE NOTES IN COMPUTER SCIENCE , 4831*, 196.

[63] *The Information and Content Exchange (ICE) Protocol*. (1998, October 27). Retrieved March 2010, from World Wide Web Consortium (W3C): http://www.w3.org/TR/NOTE-ice

[64] Theobald, A., & Weikum, G. (2002). The XXL search engine: Ranked retrieval of XML data using indexes and ontologies. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data* (p. 615). ACM.

[65] Viyanon, W., & Madria, S. (2009). A System for Detecting XML Similarity in Content and Structure Using Relational Database. *Proceeding of the 18th ACM conference on Information and knowledge management* (pp. 1197-1206). Hong Kong, China: ACM.

[66] Viyanon, W., & Madria, S. (2009). XML-SIM: Structure and Content Semantic Similarity Detection using Keys. *Proceeding of the 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2009)* (pp. 1183-1200). Vilamoura, Algarve-Portugal: Springer.

[67] Viyanon, W., Madria, S., & Bhowmick, S. (2008). XML Data Integration Based on Content and Structure Similarity Using Keys. *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:* (pp. 484-493). Springer.

[68] Weis, M. (2005). Fuzzy Duplicate Detection on XML Data. *Proceedings of VLDB 2005 PhD Workshop*, (p. 11).

[69] Weis, M., & Naumann, F. (2005). DogmatiX tracks down duplicates in XML. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 431-442). USA: ACM.

[70] Wen, L., Amagasa, T., & Kitagawa, H. (2009). An approach for XML similarity join using tree serialization. *Proceedings of the 13th international conference on Database systems for advanced applications* (pp. 562-570). Springer-Verlag.

[71] *XML Version of DBLP*. (n.d.). Retrieved May 2006, from http://dblp.uni-trier.de/xml/

[72] Yan, T., & Garcia-Molina, H. (1999). The SIFT information dissemination system. *ACM Transactions on Database Systems (TODS)* , 565.

[73] Yoshikawa, M., Amagasa, T., Shimura, T., & Uemura, S. (2001). XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology , 1* (1), 110-141.

[74] Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing , 18*, 1245.

VITA

Waraporn Viyanon was born in Bangkok, Thailand, on June 26, 1975. She completed a bachelor's degree in computer science from Srinakharinwirot University, Bangkok, Thailand, in 1997. In 2001, she received her master of science in applied computer science from Illinois State University, Normal, Illinois.

She enrolled at Missouri University of Science and Technology (formerly University of Missouri-Rolla) in 2006 to pursue a doctorate in philosophy in computer science. She received her Ph.D. from the Department of Computer Science in December 2010.