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 for the structure and content semantic similarity detection between two XML documents from heterogeneous data sources using the notion of *keys.* Comparisons with the previous systems (XDoI and XDI-CSSK) are presented to show that our new approach has a better performance by a big order of magnitude in terms of detection, false-positives and execution time.

Keywords: XML Similarity Detection, keys, clustering, matching.

1 Introduction

XML has been increasing relevance as a means for exchange information and complex data representation on the Internet [7]. As a matter of fact, different XML sources may have similar contents, but described using different tag names and structures such as bibliography data; DBLP [20] and Sigmod Record [1]. Integration of the similar XML documents from different data sources benefits users to get access to more complete and useful information.

Since XML documents encode not only structure but also data, to measure accurate similarities among them for XML document integration, it requires similarity computation on both structure and content. In most of the matching algorithms, XML documents are considered as a collection of items represented in XML tree forms. Then they are fragmented into small independent items representing objects called subtrees. Next, to find out which subtrees are similar between two XML documents, their subtree similarities are measured in terms of both structure and content. The subtree pairs having higher similarity than a given threshold are considered as matched pairs which are finally integrated into one XML document. There are recent works on XML document integration such as SLAX [12] and [3, 4, and 6]. Our earlier proposed work on XML integration techniques are XDoI [18] and XDI-CSSK [17] that outperforms SLAX [12]. However, in these two approaches, we first find XML content similarity degrees without taking its structure into account and then comparing the structure similarity later. It should be rather other way around, comparing content similarity from the two given subtrees based on their similar structures. The content similarity degrees are measured by computing common leaf-node values from a subtree in a base XML document against another subtree in the target XML document. This is a

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part II, LNCS 5871, pp. 1183–1200, 2009.

time consuming method; it would have been better to compare leaf-node values only when they have similar structures.

In this paper, our objective is to design, implement and evaluate a framework of XML integration based on XML structure and content similarity detection using keys and semantic matching (see Section 3). This framework is an improvement on our previous works (XDoI [18] and XDI-CSSK [17]) as we focus on semantics associated with the child nodes in a subtree that helps in reducing the number of subtree comparisons to be made. The contributions of this paper can be summarized as follows:

1. We proposed an improved framework for XML integration which uses some of the earlier on the clustering of XML documents into subtrees, key(s) finding and matching subtrees using key(s) from [17] and the metric of semantic similarity based on Information Content using Java WordNet Similarity Library (JWSL) from [14]. We define a new method of computing similarity between two XML documents in terms of both structure and content. The matching subtree algorithm has been implemented based on structure (path) semantic similarity.
2. We performed experiments on bibliography data sources, ACS SIGMOD Record [1] and DBLP [20] and evaluated the proposed framework by comparing it with the previous systems in order to depict how this approach gets improvement in different parameters such as similarity detection and execution time.
3. Our experimental results shows impressive improvement in terms of execution time over the earlier papers [17, 18] and the number of false positives has been reduced by 12.84% in comparison to them.

2 Related Work

Similarity detections in XML can be categorized into groups: (1) structural similarity and (2) content and structural similarity. The structural similarity detections mostly are used in document clustering and change detection. The content and structural similarity is appropriate to be employed in document integration.

There are several approaches [3, 4, 6], for structural similarity in tree-based documents based on finding the least edit-distance [22] between two documents by identifying how the first document can be edited to become the second document. There are also works on structure-oriented similarity which aimed to extract pure structural information from documents. Tree Edit Distance (TED) measures the minimum number of node insertions, deletions and updated which are required to convert one tree into another. TED assigns a cost value of 1 to each edit operation as default [3, 4]. The edit distance between two trees is the smallest cost of transforming T1 to T2. The computation of the tree edit distance is of $O(n^2 min^2(l, d))$ time and $O(n^2)$ space for a tree with n nodes, l leaves, and depth d [22].

Path similarity [11] measures the similarity of paths between two different documents. 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 are discovered; or weighted, where the nodes are weighted according to their distance from the root. The partial path similarity measures are expensive to compute as there are $n!$ mappings between the paths of two trees. They are exhaustive algorithms that produce the optimal similarity score.

Structural similarity can be compared using XML DTD by comparing a document's DTD against another document's DTD in order to identify commonalities and differences between them; however, sometimes XML DTD might not be available.

To identify the similarity between two elements, there are many approaches: (1) string matching, (2) edit distance and, (3) semantic similarity. These approaches are effective and widely used metrics for measuring similarity. String matching is to check whether the strings are identical. It is simple to implement using string matching but the result of similarity may miss some similar strings. The distance between strings s and t is the cost of the best sequence of edit operations that converts s to t. As mentioned before, edit distance is time-consuming and the similarity result may not be accurate in terms of semantic. Another approach which is in many ways similar to the edit distance is the Longest Common Subsequence (LCS) approach [2]. It finds the longest sequence of tokens common to the two strings.

Semantic similarity methods [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: edge-counting-based methods [15] and information corpus-based methods.

The information theory-based method for semantic similarity was first proposed by [16]. The similarity of two concepts is defined as the maximum of the information content of the concept that subsumes them in the taxonomy hierarchy. The information content of a concept depends on the probability of encountering an instance of the concept in a corpus. The probability of a concept is determined by the frequency of occurrence of the concept and its sub-concept in the corpus. The information content is then defined as negative the log of the probability. [9] proposed a combined method that is 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, so 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. [13] derived a theoretically well-motivated measure that is similar to Resnik's information content [16]. Lin's modification [13] consisted of normalizing by the combination of information content of the compared concepts and assuming their independence.

The well-known knowledge resource as taxonomy hierarchy is WordNet [8]. It 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. Pirror and Seco have implemented Java WordNet Similarity Library (JWSL) [14] that provides methods based on information theoretic theories of similarity.

As we know that keys are an essential part of database design; they are fundamental to data models and conceptual design. Not only semantic similarity, but using XML keys also assists in the subtree matching. XML key concept is introduced in [5]. If we could identify keys in XML documents, it would reduce the number of matchings dramatically. Since most of the XML data is data-centric; derived from the relational data model, therefore, in such cases, it is better to exploit keys to find better similarity matching subtrees.

3 Problem Statement

In this paper, we focus on the drawbacks of the previous works XDoI [18] and XDI-CSSK [17]. In these methods, the subtree similarities are computed based on the content similarity first by comparing the number of common values at the leaf-node levels without considering their structure. If the content similarity results cause multiple matchings then the structural similarity (or path similarity) is taken into account to distinguish which subtree pair is more similar. This makes the similarity computation time-consuming because all leaf nodes are compared even though they are not similar in terms of their data types and semantics.

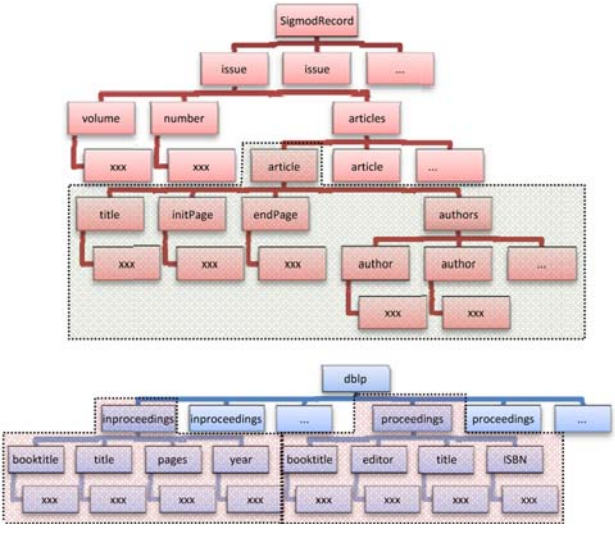


Fig. 1. Example of XML documents compared in XDoI and XDI-CSSK

Figure 1 shows XML documents structure of SigmodRecord 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 and compare all subtree pairs. It has been shown that XDI-CSSK's clustering is better than XDoI in [17] since XDI-CSSK is able to segment the XML documents into proper subtrees. For this example, the clustering points are the edges above the *article* node from SigmodRecord, and the *inproceedings* and *proceedings* nodes from DBLP. Even XDI-CSSK obtains suitable clustered subtrees in order to compare them between the two XML documents since it removes inappropriate subtree levels from the results of multiple matchings using keys, but it does not consider the document structures while doing the content similarity. The algorithms of both approaches for finding content and structure similarity are straightforward; they compare leaf nodes having the same PCDATA value. For this example, all the leaf nodes rooted by the *article* node are compared with the leaf nodes rooted by the *inproceedings* and *proceedings* nodes.

Obviously, it does not make sense to compare the value at the *title* node in the *article* subtree with the value at the *pages* or *year* nodes in the *inproceedings* subtree, since they are not similar in terms of their semantics and their data types. Therefore, this similarity computation provides no additional information while comparing these leaf nodes.

XDoI shows that the key identified can reduce the number of subtree matchings and XDI-CSSK takes advantage from the results of key matchings by analyzing the matching characteristics in order to get rid of improper subtrees so that the subtree matching does not suffer from comparing the inappropriate subtrees.

In this paper, we address the above drawbacks by considering the semantic structural similarity of leaf nodes in the distinct clustered subtrees before comparing the content at leaf nodes.

4 Our Approach

In this section, we address our approach called XML-SIM which is an improvement over XDoI and XDI-CSSK to detect similarity of two XML documents. We first explain the overall frame work of this approach and the details of each component. We then discuss the algorithm of 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 shows the overall framework of our approach.

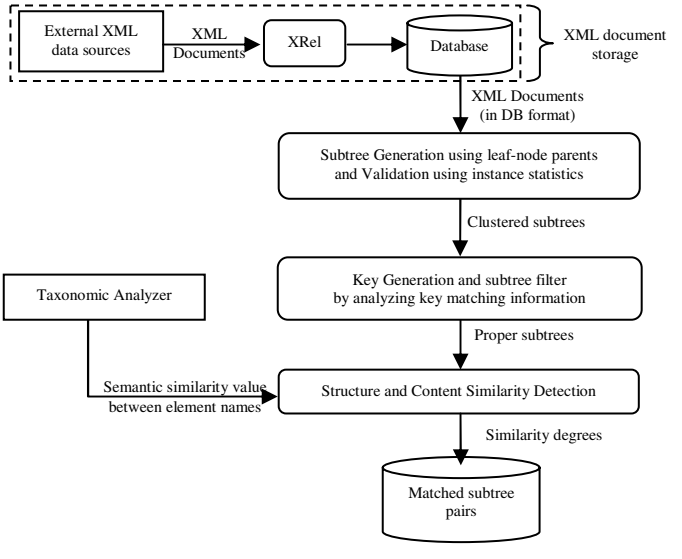


Fig. 2. XML-SIM framework

First, XML documents are stored into a relational database which increases scalability so that the memory limitation of loading very large XML trees into the main memory is not our constraint. Second, XML documents are clustered into subtrees using leaf-node parents which are verified for subtree integrity by instance statistics concept [19]. We then define XML key(s) based on a leaf-node value match for all unique node values with the same path signature. The key(s) is later used in matching subtrees with key(s). However, the consequences of the key matching may result multiple matchings since an identified key may be a part of one or more different subtrees according to Definition 6 in Section 4.2; this key-matching information can be used in the subtree filter process in order to get rid of inappropriate subtrees. At this point, we are mainly left with all the proper subtrees to be compared. The structures of defined proper subtrees are measured to find the semantic structural similarity based on the taxonomic analyzer. The semantic structural similarity is exploited in order to compare the content similarity. The content similarity is determined by comparing leaf-node values which have similar semantic structure. Finally, the system results into the best matched subtree pairs. These matched subtree pairs can then be integrated together.

4.2 Key Definitions for XML-SIM

In this section, we present the notations, definitions and the improved algorithm in order to solve the problem mentioned in Section 3.

4.2.1 XML Document Storage

First, we provide some definitions regarding XML documents and describe the storage model to store XML documents.

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]. XRel decomposes an XML document into nodes on the basis of its tree structure and stored in relational tables according to the node type, with path information from the root to each node. The basic XRel schema consists of the following four relational schemas:

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

Fig. 3. XRel's schema

The database attributes "docID", "pathID", "start", "end" and "value" represent document identifier, simple path expression identifier, start position of a region, end position of a region, and string-value respectively. The occurrence of 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 kept 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 explained in Definition 2.

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

4.2.2 Subtree Generation

In this subsection, we give definitions related to the subtree clustering and discuss 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 a child as a leaf node. This node is considered as a root of subtree 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
```

Fig. 4. SQL query for finding leaf-node parents

Definition 4: Clustering Point – *An edge e_c is an edge between nodes v_p and v_i. The edge e_c is a clustering point iff $(v_p, v_i) \in E$, where v_i is the parent of v_p and v_p.is the leaf-node parent from Definition 2. 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 indicates the place for clustering an XML tree into subtree(s).*

The clustered subtrees are categorized into two types, *simple subtree* and *complex subtree* discussed in Definitions 5 and 6 below.

Definition 5: Simple Subtree – *Given two XML document trees T_b and T_t, where T_b denotes the base tree and T_t denotes the target tree. Both T_b and T_t are clustered into k_b and k_t subtrees respectively, where $t_{bi}(1 \le i \le k_b)$ and $t_{tj}(1 \le j \le k_t)$. The subtree t_{bi} with a node set V_{bi} is a "simple subtree" iff (i) $num\_parent(v_{pi}) = 0$, (ii) $num\_parent(v_{pi}) = 1$ and parent of $v_{pi}$is NOT a leaf-node parent (from Definition 3), where v_{pi} is a leaf-node parent in the subtree t_{bi} and num_parent() is a function to count the number of parents. This condition is applied to the subtree t_{tj} as well. A simple subtree is a clustered tree with only a root and leaf*

Definition 6: Complex Subtree – *Any clustered subtrees t_i' with a node set V_i' are complex subtrees iff 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 in Figure 5. The *leafnode_parent* relation stores path signatures that have leaf-nodes and its parent path expressions. The pathids of parent paths can be retrieved from the *path* relation. The *subtree* relation keeps the clustered subtrees which are used in similarity comparison later on. Each subtree contains path information, content values at the leaf-node level and a key flag. The key flag is used to identify the leaf-node uniqueness based on the same pathid. The 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)

Fig. 5. XML-SIM's relations

4.2.3 Subtree Validation

A subtrees representing an independent object should contain nodes representing different information; it should not have only one kind of nodes. For example, in Figure 1 <authors> is the parent for two <author> nodes which are its leaf nodes. The <authors> node is considered as the root of the subtree which has two <author> nodes as its children. It is obvious that this kind of subtree does not contain any other information than <author> and therefore, it is not very useful to extract this kind of subtrees to be compared in the subtree similarity measurement.

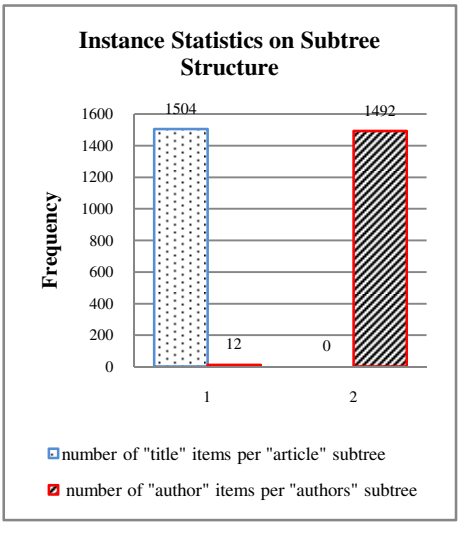


Fig. 6. Statistics on subtree structure

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

Fig. 7. Remove leaf-node parents not having one-to-one relationship

We apply the instance statistics concept based on subtree element structure [19] in order to check the relationship between the leaf-node parent element and its children's leaf-node elements whether they preserve a loose 1:1 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 the frequencies of the number of <title> elements per <article> subtree have exactly one title; on the other hand; the majority of the number of "author" occurrences per "authors" subtree is two. The relationship between <article> and <title> is called a 1:1 relationship. The leaf-node parents not having a loose 1:1 relationship with their children are removed from the *leaf-node_parent* relation using the SQL in Figure 7.

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. We identify the possible keys for the XML documents by the SQL query in Figure 8 retrieving unique values from the *text* relation that can be used to distinguish items from others.

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

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

Fig. 8. SQL query to identify leaf nodes as key(s)

The labels associated with the returned leaf nodes are considered as subtree keys. We flag "Y" in the attribute "key" on the matched records (according to their docid, pathid, and value) in the *subtree* relation.

4.2.5 Subtree Matching Using Subtree Keys

The subtree keys found in the previous step are used to match subtrees by comparing the subtrees having their leaf nodes (labels) marked as "key" and having identical values. The key matching results are stored in a temporary relation called *v_key_match*. The subtree key matching may cause multiple matchings, stored in *v_key_manymatching*, since complex subtrees contain multiple subtrees which may

have leaf nodes defined as “key”. Even though this comparison ignores the structure of the leaf nodes but its matching results can be analyzed in order to find out which subtree level is not appropriate to be compared. We analyze the matching information by examining the number of subtree matchings less than the median number of alternate keys. The intuition behind is that a complex subtree may contain a huge number of simple subtrees having alternate keys inside. This kind of complex subtrees will cause several useless matchings and are considered as improper subtrees. To eliminate the improper subtrees, we define a threshold calculated using the median number of the alternate keys. The subtrees causing the number of multiple matchings more than the median number of the alternate keys are eliminated. The results of the key matching are retrieved by the SQL query in Figure 9 and are analyzed using the SQL queries in Figure 10 (a) and (b) in order to find the proper leaf-node parents.

```
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
```

Fig. 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
```

Fig. 10. (a) SQL query for finding multiple matching over the median number of alternate keys (b) SQL query for finding proper 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.

4.2.6 Structure and Content Similarity Detection

In order to detect the right matched subtree pairs, we consider both structure and content of the base and target XML trees. First, we define the Path Semantic Similarity Degree base on the 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*;* V_l *has* n *elements.*

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* $1 \le k \le n_t$*;* n_b *and* n_t *are the number of leaf nodes in the base subtree and target subtree respectively to determine the path semantic similarity. To be able to measure the path similarity between* p_{lj} *and* p_{lk}*, we need to compare the node labels from* p_{lj} *and* p_{lk} *first.*

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 Jiang's and Resnik's information theory-based methods [9, 16] and defined as follows:*

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

The IC value is calculated by considering 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 basic intuition behind the use of the negative likelihood is that the more probable a concept is of appearing then 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 as follows:

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

Definition 9: Path Semantic Similarity Degree (PSSD) – *A path semantic similarity degree is the ratio of summation of 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 written in the equation below 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 10: Matched Path Pair (MPP) – *A matched path pair is from the maximum pair having 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 is the path expression having MPP() value greater that a given threshold* δ_p

The values of PSSD are stored into a *PathSim* table. We use the SQL query to retrieve the matched path pair as the Figure 11.

```
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
```

Fig. 11. SQL query for finding matched path pairs

At this point, all path expressions at the leaf-node levels are evaluated and selected. The selected paths from Definition 11 will be used to determine the content similarity among the subtrees.

Example: Here we illustrate an example of selecting a path pair: let's compare the subtree rooted by <article> node and the subtree rooted by <proceedings> node in the Figure 1. 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_{t1} = /proceedings/booktitle
p_{b2} = /article/initPage	p_{t2} = /proceedings/editor
p_{b3} = /article/endPage	p_{t3} = /proceedings/title
p_{b4} = /article/authors/author	p_{t4} = /proceedings/ISBN

We then distinct the node labels from both subtrees, which are <article>, <title>, <initPage>, <endPage>, <authors>, <author> and <proceedings>, <booktitle>, <editor>, <title>, <ISBN>, in order to perform NSSD computation. The results of calculating NSSD are shown in Table 2. Note that <authors> is the plural form of <author>, so we treat it as the same label.

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

1.1.1.1.1 $NSSD\left(l(v_j), l(v_k)\right)$

$l(v_k) \backslash l(v_l)$	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

PSSD for each pair of path expression is calculated. Here we illustrate how to compute $PSSD(p_{b1}, p_{t3})$:

$$avg(NSSD_{1<article>}) = \frac{NSSD(<article>,<proceedings>) + NSSD(<article>,<title>)}{2}.$$
$$= 0.529349$$

And

$$avg(NSSD_{2<title>}) = \frac{NSSD(<title>,<proceedings>) + NSSD(<title>,<title>)}{2}$$
$$= 0.692778$$
$$PSSD(p_{b1}, p_{t3}) = \frac{avg(NSSD_{1<article>}) + avg(NSSD_{2<title>})}{2}$$
$$= 0.611064$$

The same calculation is processed for all pairs of path expressions. Table 3 shows the results of all PSSD pairs and the selected path pair which is (p_{b1}, p_{t3}) or (/article/title, /pr□ceedings/title). This selected path pair will be used in the content similarity. It is possible that we can have more than one selected path pair.

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	0.665629
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 – *Each subtree* $t_{bi}(1 \le i \le k_b)$ *is evaluated against the subtrees* $t_{tj}(1 \le j \le k_t)$ *by comparing PCDATA value (content approach) based on the selected path (structure approach) to decide which subtree is the proper matched subtree pair (MSP).*

This comparison based on content and structure can be done simply using loops but it would take much more time if the number of subtrees is larger. Instead of loops, we use a SQL query to retrieve subtree pairs which returns result much faster. The resulted subtree pairs based on the same leaf node parent are intersected together in order to find the best matched subtree pair satisfying the conditions, having the same PC data content and similar structure. The algorithm of finding the matched subtree pairs is presented in Figure 12.

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
}

Fig. 12. Algorithm of retrieving matched subtree pairs

5 XML-SIM Experiment

In order to observe the efficiency and effectiveness of XML-SIM algorithm, we evaluate the algorithm by comparing it with the previous works, XDoI's and XDI-CSSK's algorithms, in terms of similarity detection, accuracy and execution time.

5.1 Experimental Setup

We conduct experiments by using Intel Core 2 Duo CPU 2.20GHz with 4GB of RAM running on Window XP Professional with Sun JDK 1.6.0_02 and Oracle Database 10g Standard Edition. We used available bibliography datasets, SIGMOD Record, 482 KB as the base document and three segmented documents of DBLP.xml, 700 KB each as 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 subtrees pairs
#1	SigmodRecord (482)	DBLP1 (679)	343
#2	SigmodRecord	DBLP2 (688)	321
#3	SigmodRecord	DBLP3 (717)	67

The actual matched subtree pairs are detected manually. These numbers are used to determine the false positives in the similarity detection of each algorithm.

5.2 Experimental Results

In this section, we show the results of our experiments comparing the clustering methods and evaluating two parameters: (1) execution time and (2) accuracy of similarity detection.

5.2.1 Evaluation on Clustering Method

To verify the effectiveness of clustering XML documents into subtrees, we show the clustering points and the number of clustered subtrees for each algorithm. In XDoI, SigmodRecord is clustered into three different levels, <issue>, <article>, <authors> as 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 filter the clustered subtrees using instance statistics and information from the key-matching. For the fragmented DBLP documents, there is no difference among these three approaches because the structure of DBLP document is shallow and has only one level defined as the clustering point. The results of clustering points and the number of clustered subtrees are shown in Table 5 (a and b).

Table 5. (a) The number of clustered subtrees based on the clustering points in SigmodRecord.xml

6	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

Table 5. (*continued*)

(b) The number of clustered subtrees based on the clustering points in DBLP1, DBLP2 and DBLP3

	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

5.2.2 Evaluation of Execution Time

Here, we evaluated the performance of the experiments in terms of how fast each algorithm is to find matching subtrees on each document pair. We ran the experiments using $\hat{o} = 0.5$ as a user defined threshold value. The threshold in XDoI and XDI-CSSK is used to measure the content similarity but in XDI-SIM it is used to evaluate the structural similarity.

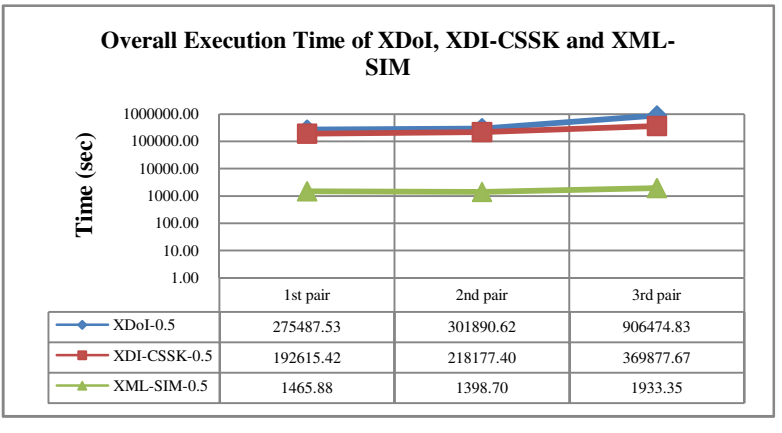
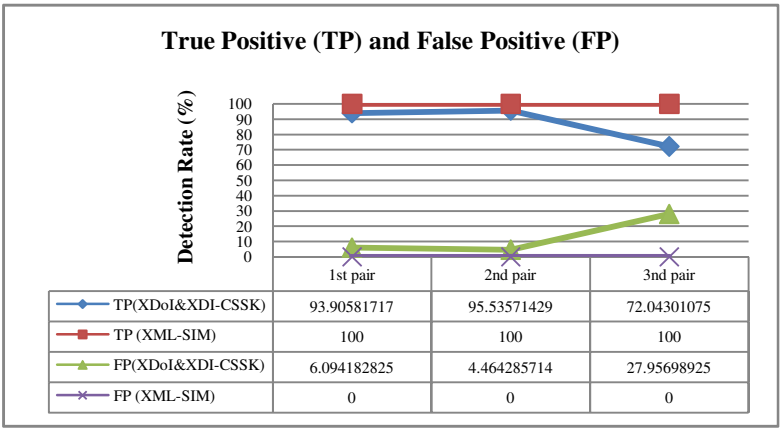


Fig. 13. Overall execution time in XDoI, XDI-CSSK and XML-SIM

Figure 13 shows the execution time of each approach in a logarithmic scale base 10; it depicts that XDI-CSSK performs better than XDoI since XDI-CSSK eliminates improper subtrees using the results from key-matching. In addition, XML-SIM dramatically outperforms both approaches. This means that the structure comparison in the early stage helps the system detect the subtree similarity faster. Note that the similarity computation on the 3rd pair takes much more time than others because the number of subtrees in the 3rd pair is higher than both pairs.



	1st pair	2nd pair	3nd pair
TP(XDoI&XDI-CSSK)	93.90581717	95.53571429	72.04301075
TP (XML-SIM)	100	100	100
FP(XDoI&XDI-CSSK)	6.094182825	4.464285714	27.95698925
FP (XML-SIM)	0	0	0

Fig. 14. Detection rate on subtree matching

5.2.3 Evaluation of Similarity Detection

Here, we evaluated the effectiveness of our our approach by determining false positives and true positives. A false positive value reflects the ratio of the number of incorrectly detected matched-subtrees and the number of actual matched subtrees and a true positive value reflects the ratio of the number of correct matched subtrees and the number of actual matched subtrees. It shows that XML-SIM outperforms XDI-CSSK [18] and XDoI [17] as it has no false positive for the three pairs of the documents as shown in Figure14. This is because we detect the semantic structural similarity at an early stage of the similarity comparison. The results from the selected path pair can also identify the similar matching structure in the other document.

6 Conclusions and Future Work

This paper presents an improved algorithm called XML-SIM based on our previous works XDoI and XDI-CSSK to detect the XML semantic similarity based on structure and content. The main improvement of this approach over others is that the content similarity is determined based on first finding the semantic-based structural similarity using semantics. This algorithm as shown by experimental evaluations outperforms XDoI and XDI-CSSK approaches in terms execution time as well as false positive rates. We aim to extend our current work to find the similarity among multiple versions of XML documents.

References

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

[2] Apostolico, A., Galil, Z.: Pattern matching algorithms. Oxford University Press, USA (1997)

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

[4] Bille, P.: Tree edit distance, alignment distance and inclusion. IT Univ. of Copenhagen TR-2003-23 Citeseer (2003)
[5] Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.C.: Keys for XML. Computer Networks 39(5), 473–487 (2002)
[6] Cobena, G., Abiteboul, S., Marian, A., Inria, R.: Detecting changes in XML documents. In: Proceedings. 18th International Conference on Data Engineering, 2002, pp. 41–52 (2002)
[7] Extensible Markup Language (XML), http://www.w3.org/XML/ (accessed, March 2006)
[8] Fellbaum, C., et al.: WordNet: An electronic lexical database. MIT press, Cambridge (1998)
[9] Jiang, J.J., Conrath, D.W.: Semantic similarity based on corpus statistics and lexical ontology. In: Proc. of Int. Conf. Research on Comp. Linguistics X, Taiwan (1997)
[10] Li, Y., Bandar, Z.A., McLean, D.: An approach for measuring semantic similarity between words using multiple information sources. IEEE Transactions on knowledge and data engineering 15(4), 871–882 (2003)
[11] Liang, W., Yokota, H.: A path-sequence based discrimination for subtree matching in approximate XML joins. In: Proceedings. 22nd International Conference on Data Engineering Workshops, 2006, pp. 23–28 (2006)
[12] Liang, W., Yokota, H.: SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes. In: IPSJ Digital Courier (J STAGE), pp. 382–392 (2006)
[13] Lin, D.: An information-theoretic definition of similarity. In: Proceedings of the Fifteenth International Conference on Machine Learning, pp. 296–304 (1998)
[14] Pirro, G., Seco, N.: Design, Implementation and Evaluation of a New Semantic Similarity Metric Combining Features and Intrinsic Information Content. In: 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, Heidelberg (2008)
[15] Rada, R., Mili, H., Bicknell, E., Blettner, M.: Development and application of a metric on semantic nets. IEEE transactions on systems, man and cybernetics 19(1), 17–30 (1989)
[16] Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: Proc. of IJCAI, pp. 448–453 (1995)
[17] Viyanon, W., Madria, S.K.: 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., Madria, S.K., Bhowmick, S.S.: XML Data Integration Based on Content and Structure Similarity Using Keys. In: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, 2008, pp. 484–493. Springer, Heidelberg (2008)
[19] Weis, M.: Fuzzy Duplicate Detection on XML Data. In: Proceedings of VLDB 2005, PhD Workshop, vol. 11 (2005)
[20] XML Version of DBLP, http://dblp.uni-trier.de/xml/ (accessed, May 2006)
[21] 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)
[22] Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM journal on computing 18, 1245 (1989)