# A Novel Approach to Measuring Structural Similarity between XML Documents

Buhwan Jeong, Daewon Lee, Hyunbo Cho, and Boonserm Kulvatunyou

**Abstract**

Measuring structural similarity between XML documents has become a key component in various applications, including XML mining, schema matching, and web service discovery, among others. This paper presents a novel structural similarity measure incorporating kernel methods into XML documents. Results on preliminary simulations show that this approach outperforms conventional ones.

Keywords: Information compatibility analysis, kernel methods, string kernel, structural similarity, XML mining

## Introduction

Nowadays, XML has been rooted as the standard means to express and exchange data among enterprise applications. Along with its explosive use, it has several bothersome obstacles including profusion, redundancy, and reproduction of similar information contents. The proper manipulation of XML content has become a main research issue both in academia and in industry. Two of the main issues involve XML formalisms [1, 2, 3] and a variety of similarity measures [1, 3, 4, 5, 6]. Most of those measures focus on the semantic/linguistic similarity between data items; in this paper, however, we focus on measures of structural similarity.

This paper proposes a novel structural similarity measure for comparison of XML documents. We base this measure on well-known kernel methods for structured data. We first introduce an interface representation to capture the structure of an XML document, and then deploy the kernel methods to manipulate that representation. We use this approach to compute measures for two examples: OAGIS BOD[1] data and with ACM SIGMOD Records.

The rest of the paper is organized as follows. Section 2 illustrates a motivating example, in which software components are replaced based on the semantic similarity between information models. Section 3 reviews string kernels. Section 4 describes our use of these kernels to compute the structural

[1] The OAGIS BOD (Business Object Document) schemas are open and standard specifications for supporting interoperable data exchange by providing enterprise/domain-neutral data structures and definitions. http://www.openapplications.org

similarity between XML documents. Section 5 includes preliminary simulation results, and Section 6 provides our concluding remarks.

## Motivating Example: Component Replacement and Selection

Consider the following common example. A company decides to replace a software component that is integrated with other software components in the enterprise. This decision may arise because the original component provider no longer exists, does not support that particular version of the software any longer, or may have a newer version that is deemed to be more powerful. It may also arise when another vendor has a better or less expensive alternative. In either case, the principal problem is to determine if the new software component is compatible with the functionality of and easily integratable with the other existing software component(s).

To find the answer to this problem, an IT manager must perform an information compatibility analysis. This analysis is complicated because, as noted above, this replacement must meet both functional and connectivity requirements. Fig.1 illustrates this situation with some particular software components. Suppose that the company has an Inventory Visibility (IV) system that is integrated already with its ERP system and has the necessary Web interfaces to exchange inventory data with its suppliers. The IV system can provide status updates to the visualization software and manage the inventory levels based on a specific inventory management policy [7].

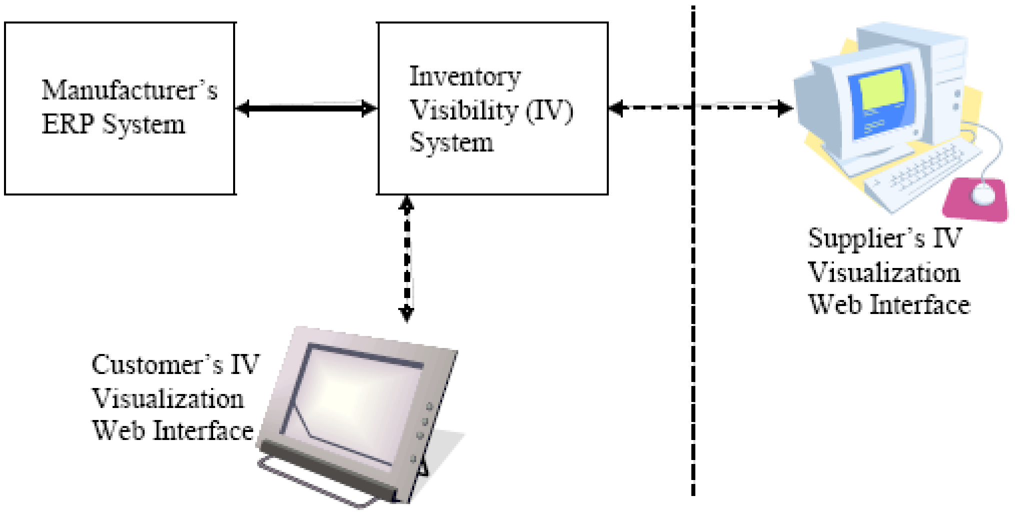

Manufacturer's ERP System
Inventory Visibility (IV) System
Supplier's IV Visualization Web Interface
Customer's IV Visualization Web Interface

Fig.1 A software component connectivity scenario

Since the ERP typically does not provide these capabilities, it is common for the ERP and the IV system to be separate software components provided by different software vendors. Therefore, an integration interface exists between the ERP and the IV systems as indicated by the bold-solid arrow connection in Fig.1. This also implies that a mapping between the corresponding information models exists. Fig.2 shows part of such a mapping. The most desirable software replacement should have an information model compatible with (or similar to) those in the IV system as well as in the ERP.

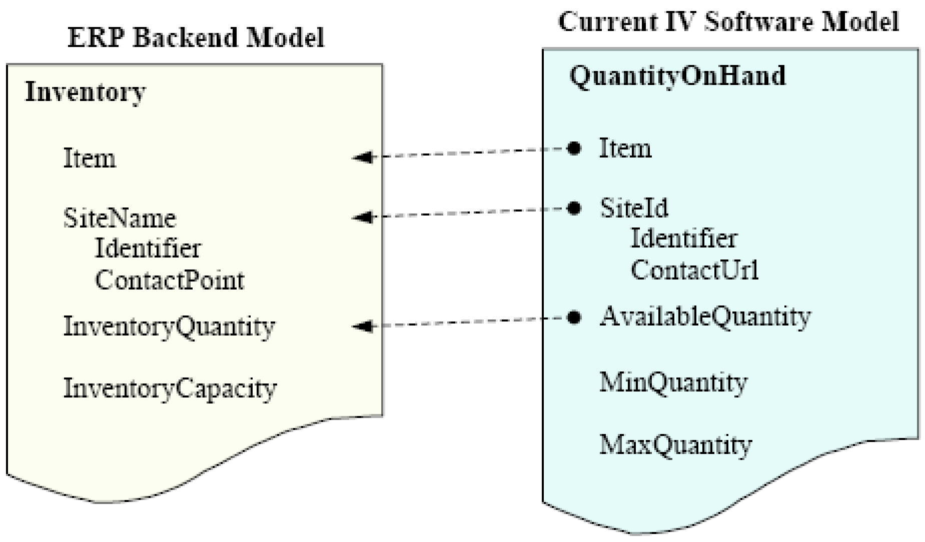


Fig.2 An exemplary mapping of data between the ERP and the IV system

## Kernel Methods for Structured Data

Kernel methods, such as support vector machines [9], use non-linear algorithms to map samples in one space ***X*** into other samples in a higher-dimensional Hilbert space ***H.*** They work very well on small problems, but often have a computational explosion for larger problems [8]. Fortunately, the so-called *kernel trick* can reduce the magnitude of the explosion by getting the scalar product implicitly computed in ***H*** when an algorithm solely depends on the inner product between vectors. Recent kernel methods for structured data employ this kernel trick to incorporate types of data other than numerical and vector data. In particular, they can now deal with string data. The following definition is critical.

**Definition 1 (String Subsequence Kernel [10])** *Let $\Sigma$ be a finite alphabet. A string is a finite sequence of characters from $\Sigma$, including the empty sequence. For string $s$ and $t$, we denote by $|s|$ the length of the string $s = s_1 \dots s_{|s|}$, and by $st$ the string obtained by concatenating them. The string $s[i:j]$ is the substring $s_i \dots s_j$ of $s$. We say that $u$ is a subsequence of $s$, if there exist indices $\mathbf{i} = (i_1, \dots, i_{|u|})$, with $1 \leq i_1 < \dots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, \dots, |u|$, or $u = s[\mathbf{i}]$ for short. The length $l(\mathbf{i})$ of the subsequence in $s$ is $i_{|u|} - i_1 + 1$. We denote by $\Sigma^n$ the set of all finite strings of length $n$, and by $\Sigma^*$ the set of all strings, i.e., $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$. We now define feature spaces $F_n = \mathbb{R}^{\Sigma^n}$. The feature mapping $\phi$ for a string $s$ is given by defining the $u$ coordinate $\phi_u(s)$ for each $u \in \Sigma^n$. We define $\phi_u(s) = \sum_{\mathbf{i}: u = s[\mathbf{i}]} \lambda^{l(\mathbf{i})}$, for some $\lambda \leq 1$. These features measure the number of occurrences of subsequences in the string $s$ weighting them according to their length. Hence, the inner product of the feature vectors for two string $s$ and $t$ gives a sum over all common subsequences weighted according to their frequency of occurrence and lengths*

$$K_n(s,t) = \sum_{u \in \Sigma^n} \langle \phi_u(s) \cdot \phi_u(t) \rangle = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}: u = s[\mathbf{i}]} \sum_{\mathbf{j}: u = t[\mathbf{j}]} \lambda^{l(\mathbf{i}) + l(\mathbf{j})}. \qquad (1)$$

Since a direct computation of these features would involve $\mathcal{O}(|\Sigma|^n)$ time and space, a recursive computation in $\mathcal{O}(n|s||t|)$ is provided in [10]. The *K*(*s*, *t*), the inner product of the feature vectors, is defined as the similarity between the strings *s* and *t* [11]. In addition, an extension to the basic string kernel is found in [12, 13], where the characters are replaced with words or syllables -- **word sequence kernel** -- and soft matching is allowed. This extension yields a significant improvement in computation efficiency for large documents.

Furthermore, one of the most critical factors to determine kernels' performance is the choice of the decay factor λ. Compared with the original string kernel, which uses the same λ for every character, [12] introduces a different λ -weighting strategy that assigns a different weight ($\lambda_c$) to each character ($c \in \Sigma$). The weighted string kernel $K^w$ of two strings *s* and *t* is defined as

$$K_n^w(s,t) = \sum_{u \in \Sigma^n} \langle \phi_u^w(s) \cdot \phi_u^w(t) \rangle = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}: u = s[\mathbf{i}]} \sum_{\mathbf{j}: u = t[\mathbf{j}]} \prod_{k=i_1}^{i_{|\mathbf{i}|}} \prod_{l=j_1}^{j_{|\mathbf{j}|}} \lambda_{s_k} \lambda_{t_l}. \qquad (2)$$

The evaluation of $K^w$ can be computed recursively using a technique similar to the one used in the original string kernel [12]. The use of different decay factors is one way of incorporating prior knowledge, such as synonymous relationships between words, into the string kernel. We discuss the determination of weights again later in the paper.

## Kernel-based Measurement of XML Structural Similarity

Our approach to computing structural similarity between XML documents using the kernel trick has two steps. First we represent tree-structured XML documents in normalized plain documents. Then, we apply the word sequence kernel to the normalized documents. We discuss these two steps in the following sections.

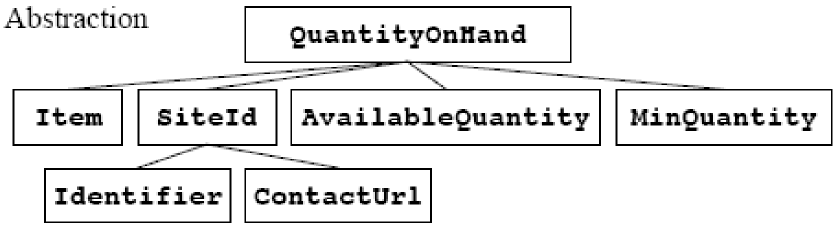
**Interface Representation**

Recall that an XML document, both XML schema and XML instance, is expressed modularly in a tree structure, which is more restricted than plain text. In such as structure, the semantics and importance of a node (including its contents) depends on its depth and order. That is, an upper node represents a more general and contextual meaning than its descendant nodes; whereas, leaf nodes often capture the specific atomic data that the XML document ultimately describes. Therefore, the interface representation should explicitly retain node order - parent-to-child and left-to-right, for example. Here, we make a hypothesis to use a sequence of node labels ordered by a depth-first traversal. The construction procedure is made up of abstraction, serialization, and normalization, as shown in Fig.3.

(A) XML Schema document

```
<xsd:element name="QuantityOnHand" type="QuantityOnHandType" />
<xsd:complexType name="QuantityOnHandType">
  <xsd:sequence>
    <xsd:element ref="Item" />
    <xsd:element ref="SiteId" minOccurs="0"/>
    <xsd:element ref="AvailableQuantity" />
    …
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="SiteId" type="SiteIdType" />
…
```

(B) Abstraction



(C) Serialization

```
QuantityOnHand Item SiteId Identifier ContactUrl AvailableQuantity …
```

(D) Normalization

```
Quantity Hand Item Site Identifier Identifier Contact Universal …
```

Fig.3 Construction of an interface representation from an XML document

The abstraction process removes unnecessary or accessory descriptions from the XML document, thereby simplifying and restructuring the original peculiar tree structure. This process is needed because XML's native DOM (Document Object Model) tree unnecessarily expresses more than the structural information. It results in an abstract tree representation that captures only the intrinsic structure to a schema when it is instantiated. The abstract tree is the most fundamental but expressive tree capturing the common structural information among various instances with the same schema. The tree is fundamental in that it disallows inclusive/duplicate structures. This means that each path, which is a sequence of element/attribute names from the root node to a leaf node[2], cannot contain element/attribute names from any other path to that node. The tree is expressive in that any content in an XML instance must be reachable by the tree. This means that the abstract tree is the collection of the longest (or most general) paths between the root and the leaf nodes[3]. For more on this process, see [14].

Second, the serialization process transforms the abstract tree representation into a sequence of words. This serialization process is the key idea to manipulating the XML's modularity because it enables us to apply the word sequence kernel without any modification. We visit every node by a depth-first traversal of the tree producing a long sentence, which is a sequence of node labels from the root node to the rightmost leaf node.

Third, the normalization process deals with the problems that each word is often a compound word comprised of several unique terms – for example, *QuantityOnHand* and *AvailableQuantity* in Fig.2. The normalization process recursively consists of (1) tokenization, which separates a compound word into atomic dictionary words; (2) lemmatization, which analyzes these words morphologically in order to find all their possible basic forms; (3) elimination, which discards meaningless stop words such as article, preposition, and conjunction, and, (4) stemming, which finds a stem form of a given inflected word [14, 15].

Take the *QuantityOnHand* schema document in Fig.2, for example. The serialization process yields *QuantityOnHand*, *Item*, *SiteId*, *Identifier*, *ContactUrl*, *AvailableQuantity*, *MinQuantity*, and *MaxQuantity.* The normalization process yields *Quantity*, *Hand*, *Item*, *Site*, *Identifier*, *Identifier*, *Contact*, *Universal*, *Resource*, *Locator*, *Available*, *Quantity*, *Minimum*, *Quantity*, *Maximum*, and *Quantity*. Note, the elimination procedures removes the preposition *On* from *QuantityOnHand;* the lemmatization process changes *Id* into *Identifier* and *Url* into *Universal*, *Resource*, and *Locator*.

**Structural Similarity Measure**

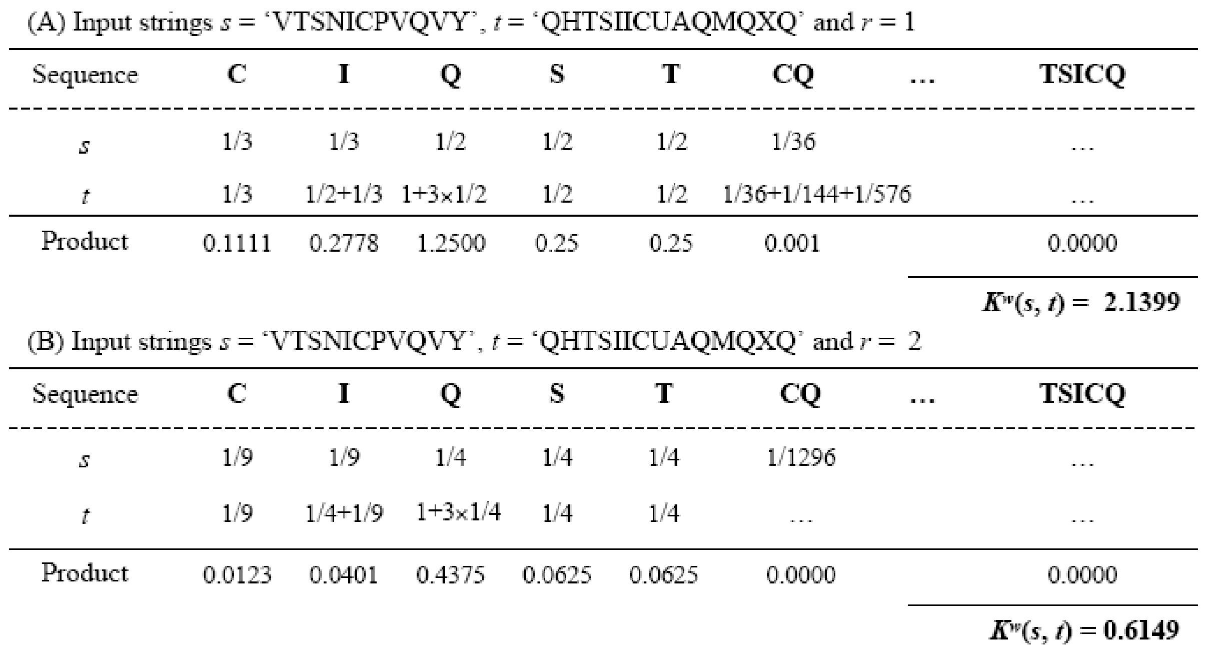
We compute structural similarity measures only for normalized documents. For two XML documents $d_1$ and $d_2$ and a kernel function $K$, we define their structural similarity as $Sim(d_1, d_2) = K(s_1, s_2)$, where $s_1$ and $s_2$ are their respective normalized strings. We use a modified, word-sequence kernel that reads a

[2] A leaf node can be either an element or an attribute, while other nodes must be elements.
[3] For XML instance documents, this abstraction step is unnecessary, but we use their DOM trees.

pair of strings, generates two feature vectors, and then calculates their inner product < , >. The final inner product is the structural similarity.

As noted above, we use assign different weights, decay factor $\lambda$, to different nodes. To make this assignment, we introduce a depth-dependent decay factor $\lambda_n = \lambda_0/depth(n)^r$, where $depth(n)$ is the depth of the node $n$ ($depth$(root) = 1) and $r >= 1$ is a relevant factor. Since, as shown in the example below, the size of inputs, length of strings is usually not a constant, the kernel value is sometimes normalized in [0, 1] by $\hat{K}(s,t) = K(s,t)/\sqrt{K(s,s) \cdot K(t,t)}$, $\hat{K}(s,t) = 1$ if and only if strings $s$ and $t$ are identical.

Take an illustrative example to compute the structural similarity between *Inventory* and *QuantityOnHand* (in Fig.2 above). For simplicity, we assign the following alphabets to stand for corresponding words: A(vailable), C(ontact), H(and), I(dentifier), L(ocator), M(inimum), N(ame), P(oint), Q(uantity), R(esource), S(ite), T(Item), U(niversal), V(Inventory), X(Maximum), Y(Capacity). Through the interface representation transformation, we get the *Inventory* document as 'VTSNICPVQVY' and the *QuantityOnHand* document as 'QHTSIICURLAQMQXQ'. The common subsequences are {C, $I^{(2)}$, $Q^{(4)}$, S, T, $CQ^{(3)}$, $IC^{(2)}$, ..., $TSICQ^{(6)}$}[4]. Accordingly, as detailed in Fig.4, their similarity is easily computed as $K^w = 2.1399$ and $K^w = 0.6149$ with respect to $r = 1$ and $r = 2$ by equation (2) with $\lambda_0 = 1$, whereas $K = 2.3699$ by equation (1) with $\lambda = 0.5$.



(A) Input strings $s$ = 'VTSNICPVQVY', $t$ = 'QHTSIICUAQMQXQ' and $r$ = 1

| Sequence | C | I | Q | S | T | CQ | ... | TSICQ |
|---|---|---|---|---|---|---|---|---|
| $s$ | 1/3 | 1/3 | 1/2 | 1/2 | 1/2 | 1/36 | | ... |
| $t$ | 1/3 | 1/2+1/3 | 1+3×1/2 | 1/2 | 1/2 | 1/36+1/144+1/576 | | ... |
| Product | 0.1111 | 0.2778 | 1.2500 | 0.25 | 0.25 | 0.001 | | 0.0000 |

**$K^w(s, t)$ = 2.1399**

(B) Input strings $s$ = 'VTSNICPVQVY', $t$ = 'QHTSIICUAQMQXQ' and $r$ = 2

| Sequence | C | I | Q | S | T | CQ | ... | TSICQ |
|---|---|---|---|---|---|---|---|---|
| $s$ | 1/9 | 1/9 | 1/4 | 1/4 | 1/4 | 1/1296 | | ... |
| $t$ | 1/9 | 1/4+1/9 | 1+3×1/4 | 1/4 | 1/4 | ... | | ... |
| Product | 0.0123 | 0.0401 | 0.4375 | 0.0625 | 0.0625 | 0.0000 | | 0.0000 |

**$K^w(s, t)$ = 0.6149**

Fig.4 Exemplary structural similarity computation using the proposed kernel method with relevant

[4] Numbers in parentheses indicate the number of possible occurrences.

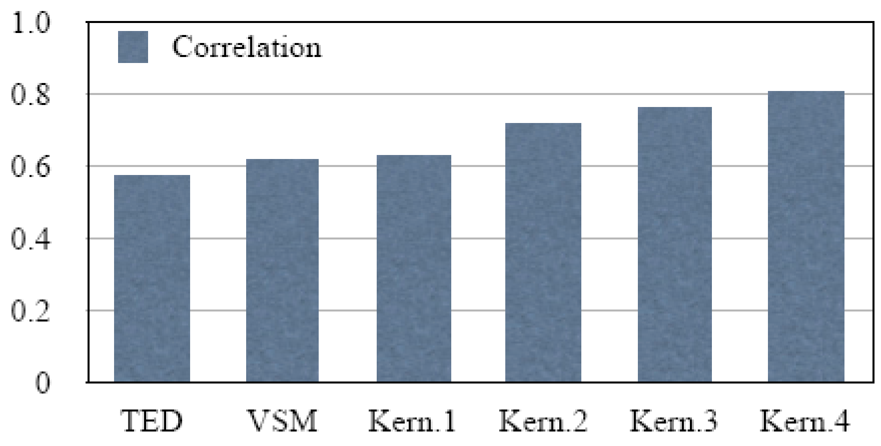
factor $r = 1$ and $r = 2$

## Preliminary Experiments

To evaluate the proposed method, we performed experiments with XML schema documents from OAGIS. We designed two types of experiments. The first was just to show that the proposed method is well suited to human judgment; the second was to verify that the proposed similarity measures could discriminate valuable information from less related information in the perspective of information retrieval.

For the first experiment, we randomly selected 200 pairs of CC's (Core Components) and let four human experts (based on their own domain and linguistic knowledge) score every pair to assign their degree of relatedness in [0, 1]. We implemented four algorithms -- TED (Tree Edit Distance)[5]; VSM by means of cosine of the angle; kernels both with a fixed penalty (i.e., $\lambda_n = c$) and with a variant penalty (i.e., $\lambda_n = f(\lambda_0, \mathit{depth}(n), r)$). The experimental result is depicted in Fig.5 in terms of correlation with the experts' average score. 'Kern.1' and 'Kern.2' implemented a fixed weighting scheme with to $\lambda = 1$ and ½ respectively. 'Kern.3' and 'Kern.4' implemented a variable weighting scheme with relevant factors $r = 1$ and 2 respectively.

As shown in the figure, the kernel methods outperform the conventional measures, TED and VSM. Although VSM is a special type of kernel methods, the proposed ones give better performance because they preserve the parent-child relationship between elements in XML documents. On other words, the bag-of-words model, VSM, gives the same importance between, for example, the root node and a leaf node. It is also noted that the proposed depth-dependent $\lambda$-weighting gives a more accurate measure than the fixed one does.



[5] A state-of-the-art similarity measure for tree structures [16].

Fig.5 Correlation between human judgment and various structural similarity measures -- TED (Tree Edit Distance), VSM with cosine of the angle, and Kernel-based measures

The second experiment was a mapping test that evaluates whether mappings established by an algorithm are correct compared with true mappings. We configured four experiment sets, each of which consisted of two data sets having 10 CC's and 20 CC's. Between the two data sets, an algorithm and human operators selected no more than 10 plausible mappings[6]. Then, we compared the selections using three widely-used performance metrics: Precision, Recall, and F-Measure. Let **A**$ be a set of alignments mapped by an algorithm, and **T** be the set of true mappings by human experts. Then the metrics are defined as follows: *Precision* = xx, *Recall* = yy, and *F-Measure* = xy. The experimental results are depicted in Fig.6. Same as the first experiment, the kernel-based measures give better performance than TED- and VSM-based ones do.

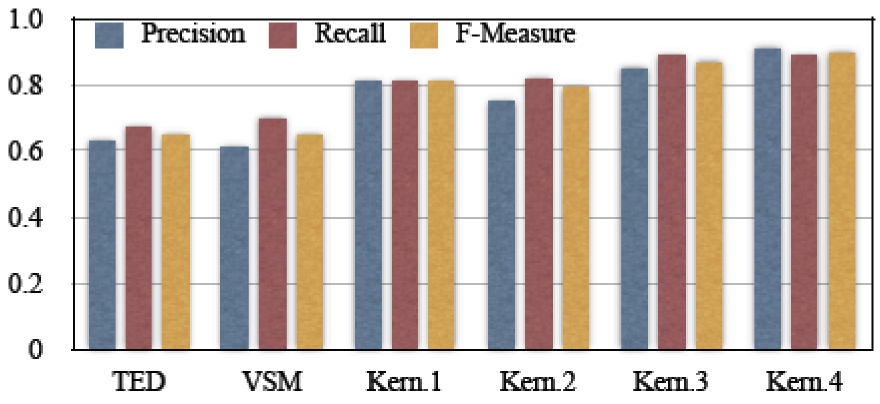
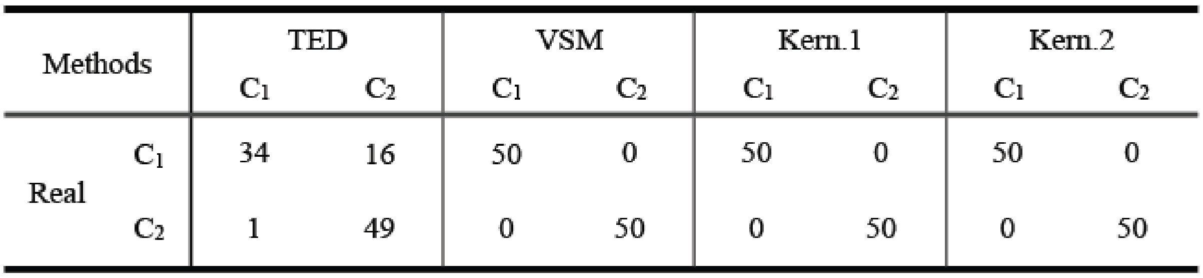


Fig.6 Precision, Recall and F-measure.

We conducted additional experiments with XML instance documents from ACMSIGMOD Records. We prepared two groups of XML instances. Each group had 50 documents randomly selected, but conforming to the same DTD (Document Type Definition). The two DTDs shared several common element definitions. Conforming to the same DTD apparently means its instances are structurally similar. We tried to discriminate the documents using the PAM (Partitioning Around Medoids) algorithm [17] with ten replications. The clustering results are depicted in Table 1, in which the low structural diversity among documents, particularly in the second group, makes them well-separated, except TED. Another implication from the result is the interface representation is a good means to express XML's tree structure. It should be noted the experiment does not care about contents in those documents, but their structure only. For all that, the proposed kernel-based measures require a significant modification to reduce computation time for large datasets.

[6] The humans selected the true mappings; the algorithm selected the test mappings.

Table 1. Clustering results for ACMSIGMOD Record: A comparison matrix



| Methods | | TED | | VSM | | Kern.1 | | Kern.2 | |
|---|---|---|---|---|---|---|---|---|---|
| | | $C_1$ | $C_2$ | $C_1$ | $C_2$ | $C_1$ | $C_2$ | $C_1$ | $C_2$ |
| Real | $C_1$ | 34 | 16 | 50 | 0 | 50 | 0 | 50 | 0 |
| | $C_2$ | 1 | 49 | 0 | 50 | 0 | 50 | 0 | 50 |

## Conclusion

This paper presented a novel approach to compute the structural similarity between XML documents by incorporating a modified string kernel. After introduction of kernel methods, we proposed an interface representation for XML documents and a λ-weighted word sequence kernel for structural similarity computation. The experimental results showed that the proposed kernel-based measure outperforms state-of-the-art approaches (i.e., TED and VSM). In particular, the research output helps web services to be discovered, selected, and composed when those activities are performed based on the message type. Moreover, we also expect that the result of this research will improve significantly the performance of many XML-based applications including XML-document clustering and classification, schema matching, XML message mapping, and ontology reconciliation.

### Disclaimer

Certain commercial software products are identified in this paper. These products were used only for demonstration purposes. This use does not imply approval or endorsement by NIST, nor does it imply that these products are necessarily the best available for the purpose.

### References

1. Flesca, S., Manco, G., Masciari, E., Pontieri, L., Pugliese, A.: Fast detection of XML structural similarity. IEEE Transactions on Knowledge and Data Engineering 17(2) (2005)

2. Yang, J., Cheung, W., Chen, X.: Learning the kernel matrix for XML document clustering. In: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05), Washington, DC., IEEE Computer Society (2005) 353–358

3. Lee, J., Lee, K., Kim, W.: Preparations for semantics-based XML mining. In: Proceedings of IEEE International Conference on Data Mining (ICDM2001). (2001) 345–352

4. Nierman, A., Jagadish, H.: Evaluating structural similarity in XML documents. In: Proceedings of the 5th International Workshop on the Web and Database (WebDB2002). (2002)

5. Shvaiko, P., Euzenat, J.: A survey of scham-based matching. Journal of Data Semantics IV 3730 (2005) 14–171

6. Jeong, B., Kulvatunyou, B., Ivezic, N., Cho, H., Jones, A.: Enhance reuse of standard ebusiness XML schema documents. In: Proceedings of International Workshop on Contexts and Ontology: Theory, Practice and Application (C&O'05) in the 20th National Conference on Artificial Intelligence (AAAI'05). (2005)

7. Ivezic, N., Kulvatunyou, B., Frechette, S., Jones, A., Cho, H., Jeong, B.: An interoperability testing study: Automotive inventory visibility and interoperability. In: Proceedings of e-Challenges. (2004)

8. Muller, K., Mika, S., Ratsch, G., Tsuda, K., Sch¨olkopf, B.: An introduction to kernel-based learning algorithms. IEEE Transactions on Neural Networks 12(2) (2001) 181–201

9. Kobayashi, M., Aono, M. In: Vector Space Models for Search and Cluster Mining. Springer-Verlag New York, Inc. (2003) 103–122

10. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. Journal of Machine Learning Research 2 (2002) 419–444
11. Vert, J., Tsuda, K., Sch¨olkopf, B. In: A Primer on Kernel Methods. MIT Press, Cambridge, MA (2004) 35–70

12. Saunders, C., Tschach, H., Shawe-Taylor, J.: Syllables and other string kernel extensions. In: Proceedings of the 19th International Conference on Machine Learning (ICML'02). (2002)

13. Cancedda, N., Gaussier, E., Goutte, C., Renders, J.: Word-sequence kernels. Journal of Machine Learning Research 3 (2003) 1059–1082

14. Jeong, B.: Machine Learning-based Semantic Similarity Measures to Assist Discovery and Reuse of Data Exchange XML Schemas. PhD thesis, Department of Industrial and ManagementEngineering, Pohang University of Science and Technology (2006)

15. Willett, P.: The porter stemming algorithm: Then and now. Electronic Library and Information Systems 40(3) (2006) 219–223

16. Zhang, Z., Li, R., Cao, S., Zhu, Y.: Similarity metric for XML documents. In: Proceedings of Workshop on Knowledge and Experience Management (FGWM2003). (2003)

17. Reynolds, A., Richards, G., Rayward-Smith, V.: The application of k-medoids and PAM to the clustering of rules. In: Intelligent Data Engineering and Automated Learning (IDEAL 2004). Volume LNCS 3177. (2004) 173–178