International Journal of Innovations in Engineering and Technology (IJIET)
http://dx.doi.org/10.21172/ijiet.223.02

Clustering XML without DTD for Similarity and Dissimilarity Calculation

Hsu-Kuang Chang
I-Shou University
No.1, Sec. 1, Syuecheng Rd.
Dashu Township, Kaohsiung,Taiwan

Abstract- XML documents on the web are often found without DTDs, particularly when these documents have been created from legacy HTML. Yet having knowledge of the DTD can be valuable in querying and manipulating such documents. Recent work (cf. [1]) has given us a means to (re-)construct a DTD to describe the structure common to given set of document instances. However, given a collection of documents with unknown DTDs, it may not be appropriate to construct a single DTD to describe every document in the collection. Instead, we would wish to partition the collection into smaller sets of "similar" documents, and then induce a separate DTD for each such set. It is this partitioning problem that we address in this paper. Given two XML documents, how can one measure structural (DTD) similarity between the two? We develop a dynamic programming algorithm to find this distance for any pair of documents. We validate our proposed distance measure experimentally. Given a collection of documents derived from multiple DTDs, we can compute pair-wise distances between documents in the collection, and then use these distances to cluster the documents.

Keywords – DTD, XML, LED, TED

I. INTRODUCTION

The Extensible Mark-up Language (XML) is seeing increased use, and promises to fuel even more applications in the future. In [1] the authors provide a method to automatically extract a DTD for a set of XML documents. They provide several benefits for the existence of DTDs. An XML document can be modeled as an ordered labeled tree [2]. There is considerable previous works on finding edit distances between trees [3–6, 7–11]. Most algorithms in this category are direct descendants of the dynamic programming techniques for finding the edit distance between strings [12]. The basic idea in all of these tree edit distance algorithms is to find the cheapest sequence of edit operations that can transform one tree into another. There are several other approaches that allow insertion and deletion of single nodes anywhere within a tree [8-11]. We account for this by introducing edit operations that allow for the cutting and pasting of whole sections of a document. Using our resulting pair-wise distance measure, we show that standard clustering algorithms do very well at pulling together documents derived from the same DTD.

II. PREPARATION FOR SEMANTIC-BASED XML DOCUMENT

In this section, we first introduce the pre-processing steps for the incorporation of hierarchical information in encoding the XML tree's paths. It is based on the preorder tree representation (PTR) [13] and will be introduced after a brief review of how to generate an XML tree from an XML document. We then describe dynamic programming mining approach to compute the similarity between two sets of encoded paths, i.e., two XML documents. To do so, we have to first go through the following five preprocessing steps for XML document. The five preprocessing steps are conversion, path extraction, nested and duplicated path removal, similar element identification and transformation, path encoding.

A. XML Document Conversion –

There are essentially two programming APIs for processing XML: SAX (Simple API for XML) and DOM (Document Object Model). DOM treats a XML document conceptually as a tree. It provides an API that allows a programmer to add, delete or edit nodes within the tree. The DOM is a collection of Recommendations maintained by the W3C (World Wide Web Consortium) [14]. We use JDOM to convert the XML document to tree format. The values of the elements in the tree are not considered here and only the structural information will be passed to the subsequent steps. The XML's hierarchical structure can be represented by a labeled rooted tree [14]. The XML tree in Figure 1 can be presented by Prefix String Pattern ($_{depth}NodeName_{Order}$) Encoding. Finally, the XML tree in Figure can be further use the adjacent linked-list tnode structure where $_{d}Node_{O}$ d is the node depth and o is the visiting order in preorder traversing in the xml tree as shown in the Table 1.

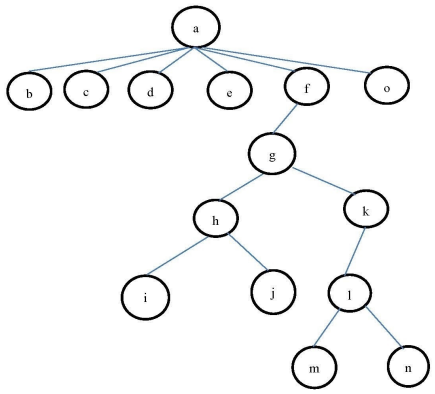

Figure 1 Simplified XML tree

Table 1 XML Tree in Adjacent List model

	tnode	Child nodes
0	${}_0a_1$	${}_1b_2 \rightarrow {}_1c_3 \rightarrow {}_1d_4 \rightarrow {}_1e_5 \rightarrow {}_1f_6 \rightarrow {}_1o_{15}$
1	${}_1b_2$	Nil
2	${}_1c_3$	Nil
3	${}_1d_4$	Nil
4	${}_1e_5$	Nil
5	${}_1f_6$	${}_2g_7$
6	${}_2g_7$	${}_3h_8 \rightarrow {}_3k_{11}$
7	${}_3h_8$	${}_4i_9 \rightarrow {}_4j_{10}$
8	${}_4i_9$	Nil
9	${}_4j_{10}$	Nil
10	${}_3k_{11}$	${}_4l_{12}$
11	${}_4l_{12}$	${}_5m_{13} \rightarrow {}_5n_{14}$
12	${}_5m_{13}$	Nil
13	${}_5n_{14}$	Nil
14	${}_1o_{15}$	Nil

B. DFS_Prefix_Encoding Search XML tree –

We used depth-first search (DFS) technique intended to transform XML tree into a prefix pattern sequence. In order to perform such a transformation, the nodes of the XML tree first have to be mapped into identifiers. Then each identifier is associated with its depth in the tree. Finally a depth-first exploration of the tree will give the corresponding prefix pattern. The DFS_Prefix_Encoding algorithm is shown in Table 2 and prefix pattern tree of XML shown in Figure 1 should be as the result ${}_0a_1$ ${}_1b_2$ ${}_1c_3$ ${}_1d_4$ ${}_1e_5$ ${}_1f_6$ ${}_2g_7$ ${}_3h_8$ ${}_4i_9$ ${}_4j_{10}$ ${}_3k_{11}$ ${}_4l_{12}$ ${}_5m_{13}$ ${}_5h_{14}$ ${}_1o_{15}$ where ${}_d\text{Node}_O$ d is the node depth and o is the visiting order in preorder traversing. Once the whole set of prefix pattern (corresponding to the XML documents of a collection) is obtained, the pair-wised XML document distance is able to calculate by dynamic programming.

Table 2 DFS_Prefix_Encoding Algorithm

DFS_Prefix_Encoding Algorithm	
1.	for each xml tree $x_{i=1\sim n}$ in adjacent-list
2.	call DFS_Prefix_Encoding(x_i,v_0)
3.	
4.	Procedure DFS_Prefix_Encoding(x_i,v)
5.	visited(v) ← 1
6.	for each vertex w adjacent to v do

7.	if visited(w)=0 then
8.	call DFS_Prefix_Encoding(x_i,w)
9.	end DFS_Prefix_Encoding
10.	

III. DYNAMIC PROGRAMMING TREE EDIT DISTANCE TED

A. Tree-Edit Transformation operations

Our algorithm for calculating the tree edit distance between structural summaries of root order- label trees that represent XML documents uses a dynamic programming algorithm. In order to transform one source tree T_1 of preorder x[1..m] to a target tree T_2 of preorder y[1..n],we can perform various transformation operations. Our goal is, given tree T_1 and T_2, to produce a series of transformations that change T_1 to T_2. Initially, i=j=1. We are required to examine every node in T_1 during the transformation, which means that at the end of the sequence of transformation operations, we must have i =m+1.

There are five transformation operations:

- Copy (↘)
  m_1 = c[i-1, j-1]+cost(copy) if x[i].label=y[j].label and x[i].depth=y[j].depth ,or ∞ otherwise.
- Replace (↖)
  m_2 = c[i-1, j-1]+cost(replace) if x[i].label≠y[j].label and x[i].depth=y[j].depth ,or ∞ otherwise.
- Twiddle (~)
  m_3 = c[x, y -1] + $c_i(T_2[y])$; if ((x, y-1), (x, y)) ∈ G (the distance of (x; y)'s top node in G plus the cost to insert T_2[y]), or ∞ otherwise.
- Delete (↑)
  m_4 = c[i-1, j] + $c_d(T_1[x])$, if ((i-1, j), (i, j)) ∈ G (the distance of (i, j)'s left node in G plus the cost to delete T_1[x]), or ∞ otherwise.
- Insert (←)
  m_5 = c[i, j -1] + $c_i(T_2[y])$, if ((i, j-1), (i, j)) ∈ G (the distance of (i; j)'s top node in G plus the cost to insert T_2[y]), or ∞ otherwise.

C[**i , j] = min(m_1,m_2,m_3, m_4,m_5),** and the corresponding operation puts into the op[i,j] table.

$$op[i,j]=\begin{cases} COPY\ or \\ REPLACE\ or \\ TWIDDLE\ or \\ DELETE\ or \\ INSERT \end{cases}$$

$$op[i,j]=\begin{cases} COPY\ or \\ REPLACE\ or \\ TWIDDLE\ or \\ DELETE\ or \\ INSERT \end{cases}$$

B. Example of Tree Edit Distance (TED)

Given two xml-tree x[1..m] and y[1..n] and set of transformation-operation costs, the edit distance from x to y is the cost of the least expensive operation sequence that transforms x to y. We use a dynamic-programming algorithm

that finds the edit distance from x[1..m] to y[1..n] and prints an optimal operation sequence, also analyze the running time and space requirements of our algorithm.

Example

The Figure 2 shows two xml trees T_i and T_j which we took feature extraction firstly, and calculates the distance between them.

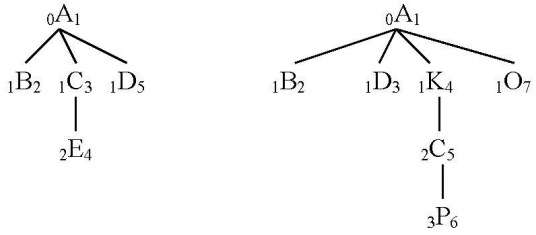

Figure 2 XML tree T_i and T_j

We calculate the distance between T_i and T_j using TED(T_i,T_j) algorithm and the result as following Table 3 shown.

Table 3 The distance between T_i and T_j using TED(T_i,T_j)

XMLs	T_j	$_0A_1$		$_1B_2$		$_1D_3$		$_1K_4$		$_2C_5$		$_3P_6$		$_1O_7$	
T_i	0	1		2		3		4		5		6		7	
$_0A_1$	1	1	↘	2	←	3	←	4	←	5	←	6	←	7	←
$_1B_2$	2	2	↑	2	↘	3	↖	4	↖	5	←	6	←	7	←
$_1C_3$	3	3	↑	3	↖	3	↖	4	↖	5	←	6	←	7	←
$_2E_4$	4	4	↑	4	↑	4	↑	5	↑	5	↖	6	←	7	←
$_1D_5$	5	5	↑	5	↖	5	↖	5	↖	6	←	7	←	7	↖

↘(copy) , ↖(replace), ↑ (delete), ←(insert)

Like longest common subsequence (LCS), our pseudo-code fills of the Table 3 in row-major order, i.e., row-by-row from top to bottom, and left to right within each row. Column-major order (column-by-column from left to right, and top to bottom within each column) would also work. Along with the *c*[*i, j*] table, we fill in the table *op*[*i, j*], holding which operation was used. To reconstruct this sequence, we use the *op* table returned by Tree-Edit-Distance. TED Operation-Print, the procedure OP-PRINT *(op, i, j)* reconstructs the optimal operation sequence that we found to transform *Xi* into *Yj* . The base case is when $i = j = 0$. The first call is OP-PRINT*(op,m, n)*.

Finally, we got the following operations which transform Ti xml tree into Tj xml tree.

Replace(T_i[1], A) /* or Copy (T_i[1],A) */
Insert(T_j[2],Ti[1],1)
Replace(T_i[2],D)
Replace(T_i[3],K)

Replace(T_i[4],C)
Insert(T_j[6],Ti[4],1)
Replace(T_i[5],O)

Also, those of the differences of two xml trees are calculated as the following:

$$\frac{7(update\cos t)}{5(delete\cos t)+7(insert\cos t)} = 0.58 \text{ dissimilarity}$$

IV. EXPERIMENTAL EVALUATION

The goal of our work is to find documents with structural similarity, that is, documents generated from a common DTD. We apply a standard clustering algorithm based on the distance measures computed for a given collection of documents with known DTDs. For any choice of distance metric, we can evaluate how closely the reported clusters correspond to the actual DTDs. The experiments were conducted as follows. The following five DTDs were downloaded from ACM's SIGMOD Record homepage[15]: OrdinaryIssuePage.dtd, ProceedingsPage.dtd, SigmodRecord.dtd , Index.dtd and IndexTerm.dtd We also downloaded the XML document generator from IBM's homepage[16]. This generator accepts the above DTDs as input and creates the sets of XML documents for simulations. Based upon five sets of XML documents with similar characteristics, their tree-edit-distance were computed, analyzed and reported as follows. We use the formula to compare pair-wise xml trees similarity $Sim(T_i,T_j)=1-TED(T_i,T_j)+Matched-Unmatched(T_i,T_j)$, and

$$Matched-Unmatched(T_i,T_j)=\frac{1}{N+1}\left(\sum_{t=1}^{N}\frac{1}{N_t}\sum_{p=1}^{N_t}\frac{m_{t,p}-c_{t,p}}{M_{t,p}}\right)$$

, where the Matched-Unmatched is difference sum of xml tree T_i and T_j in the common matched and common unmatched elements, and

N is total number of level-1 subtree,
N_t is total number of the paths in the t^{th} subtree,
$M_{t,p}$ is number of elements in the $(t,p)^{th}$ path,
$m_{t,p}$ is number of common elements (maximal sequential pattern),
$c_{t,p}$ is sum of the common unmatched element in the $(t,p)^{th}$ path.

So the difference between T_i and T_j in the Table 4 can be as followed:

$$\frac{7(update\cos t)}{5(delete\cos t)+7(insert\cos t)} = 0.58 \text{ dissimilarity (~ 0.42 similarity)}$$

, and the $Matched-Unmatched(T_i,T_j)=\frac{1}{N+1}\left(\sum_{t=1}^{N}\frac{1}{N_t}\sum_{p=1}^{N_t}\frac{m_{t,p}-c_{t,p}}{M_{t,p}}\right)$

$$=\frac{1}{5}\left(\frac{1}{1}*\frac{2}{2}+\frac{1}{1}*\frac{1-2}{3}+\frac{1}{1}*\frac{1-3}{4}\right)=\frac{1}{5}\left(1-\frac{1}{3}-\frac{2}{4}\right)=\frac{1}{30},$$

$$Sim(T_i,T_j)=1-TED(T_i,T_j)+Matched-Unmatched(T_i,T_j)=1-0.58+\frac{1}{30}=0.4533$$

A. Similarity documents of same DTDs

We show the similarity between the first document OrdinaryIssuePage as the base document, the 2^{nd} , 3^{rd} , 4^{th} , 5^{th} , and 6^{th} as the query document. Figure 3 shows the similarity between the first document OrdinaryIssuePage as the base document and the query document 2,3,4,5 and 6.

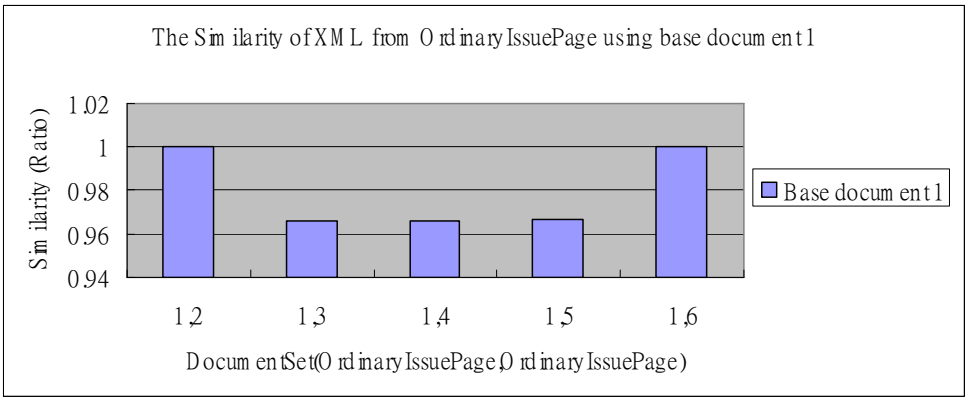

Figure 3. Similarity base-document 1 with query documents 2-6

We also compare our proposed method with Lee et al.'s method and PTR+ES method as shown on the Figure 4. It can be seen that the similarity values obtained by the proposed methods, i.e., TED, are pretty similar to those of Lee et al.'s and PTR+ES method. On the Figure 4 shows the ratio similarity of the DocumentSet(base,x)=(1,2) which uses the 1st ordinaryIssuePage as base and the 2nd OrdinaryIssuePage as query document, DocumentSet(base,x)=(1,5), DocumentSet(base,x)=(2,5), and DocumentSet(base,query)=(3,4), are better than the Lee et al.'s and PTR+ES method's similarity ratio.

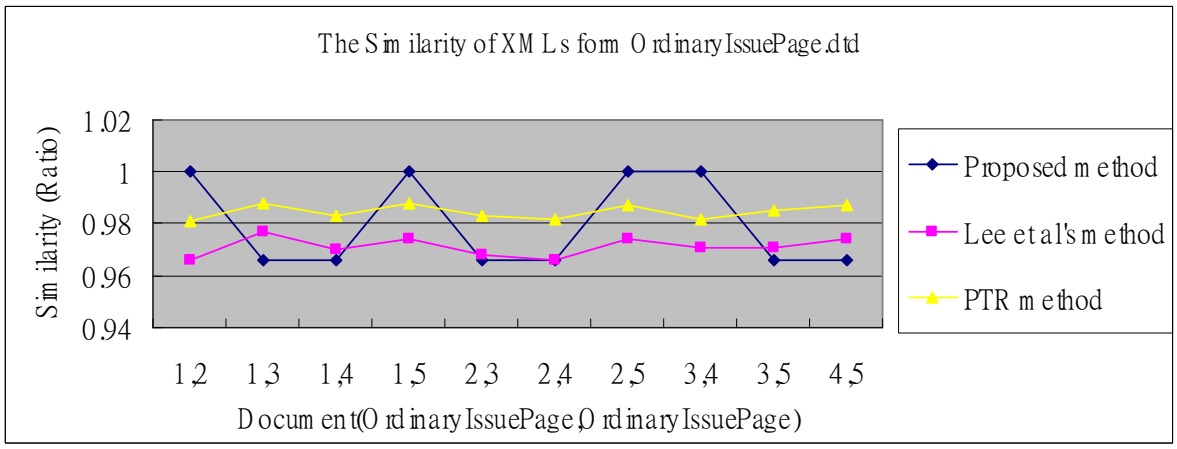

Figure 4. Comparing Similarity with different methods

B. Similarity documents of different DTDs

In this experiment, the similarities between documents of different DTDs were analyzed. Figures 5~6 show the results of heterogeneous XML document similarity. The XML documents from OrdinaryIssuePage.dtd were adopted as the base documents while those from ProceedingsPage.dtd , SigmodRecord.dtd and index.dtd were used as query documents. The experimental results are shown in Figure 5 where DocumentSet(base,x,y,z) is used to denote the similarities between document base from OrdinaryIssuePage.dtd (the 3rd document) and document x from ProceedingsPage(the 1st document ~ the 4th document) and between document base and document y form SigmodRecord.dtd and between document base and document z form index.dtd. As the XML documents come from different DTDs, this is called heterogeneous XML document similarity.

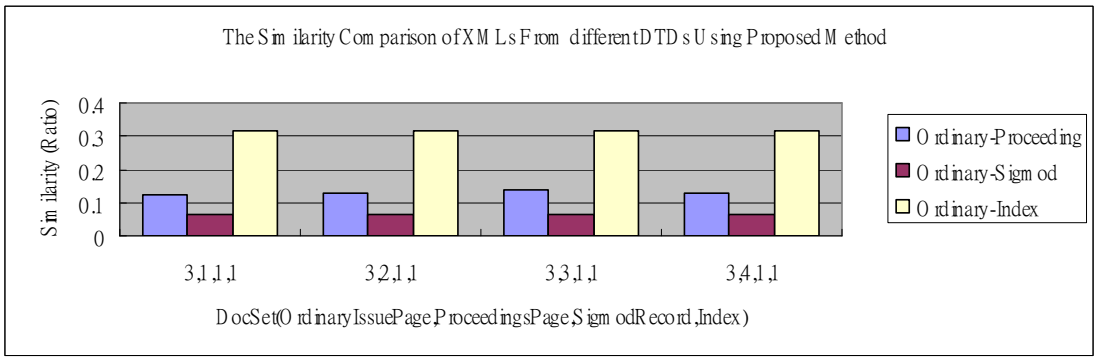

Figure 5. DocumentSet (the 3rd Ordinary as base, Proceeding, Sigmod, Index)

Figure 6 shows that DocumentSet(base,x,y,z) is used to denote the similarities between the 2nd document as base from OrdinaryIssuePage.dtd (the 2nd document) and document x from ProceedingsPage (the 1st document ~ the 4th document) and between document base and document y form SigmodRecord.dtd and between document base and document z form index.dtd.

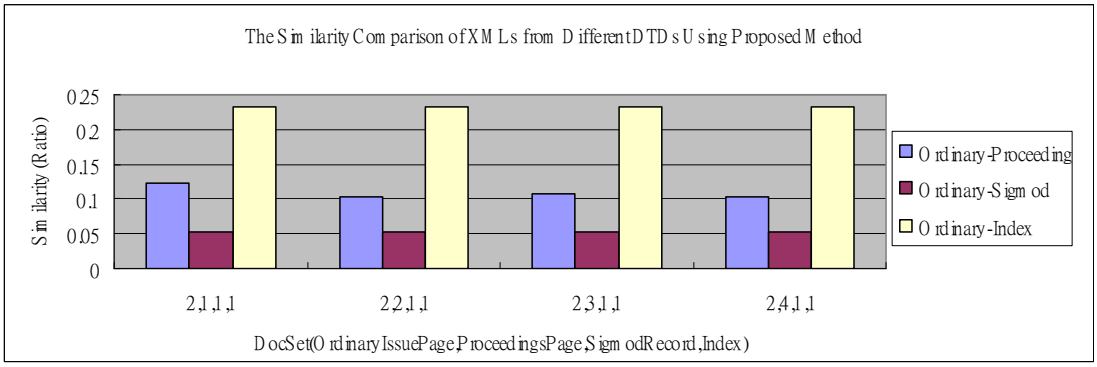

Figure 6. DocumentSet (the 2nd Ordinary as base, Proceeding, Sigmod, Index)

V. CONCLUSION

For efficiently serving versatile queries, a new XML data representation referred to as Prefix String-Pattern Encoding (PSPE) has been presented in this paper. PSPE reserves level and path depth of XML paths, the semantic information enables the inference of deriving XML path relationship. By using the algorithm TED is to find documents with structural similarity, that is, documents generated from a common DTD. We prepare for clustering based on the distance measures computed for a given collection of documents with known DTDs, and give a satisfied experiment result.

REFERENCES

[1] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim, Xtract: A system for extracting document type descriptors from XML documents. In *Proc. of ACM SIGMOD*, pages 165–176, 2000.
[2] World Wide Web Consortium. The document object model http://www.w3.org/DOM/.
[3] S. Chawathe, Comparing hierarchical data in extended memory. In *Proc. of VLDB*,pages 90–101, 1999.
[4] S. Chawathe , H. Garcia-Molina, Meaningful change detection in structured data. In *Proc. of ACM SIGMOD*, pages 26–37, 1997.
[5] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, Change detection in hierarchically structured information. In *Proc. of ACM SIGMOD*, pages 493–504, 1996.
[6] Gregory Cobena, Serge Abiteboul, and Amelie Marian, Detecting changes in XML documents, In *Proc. of ICDE*, 2002.
[7] S. Selkow, The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, December 1977.
[8] D. Shasha and K. Zhang, Approximate tree pattern matching, In *Pattern Matching in Strings, Trees and Arrays*, chapter 14, Oxford University Press, 1995.
[9] K. C. Tai, The tree-to-tree correction problem. *Journal of the ACM*, 26:422–433, 1979.
[10] J. Wang, K. Zhang, K. Jeong, and D. Shasha, A system for approximate tree matching, *IEEE TKDE*, 6(4):559–571, 1994.
[11] K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245–1262, December 1989.

[12] V. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, 6:707–710, 1966.
[13] Sedgewick R (1996) Chapter 5 trees, an introduction to the analysis of algorithms. Addison-Wesley, pp 221–298.
[14] World Wide Web Consortium. The document object model. http://www.w3.org/DOM/.
[15] ACM SIGMOD Record home page [http://www.acm.org/sigmod/record/xml]
[16] IBM's XML Generator homepage [http://www.alphaworks.ibm.com]