Improve Query Performance by Clustering XML Documents

Wang Lian David W. Cheung Nikos Mamoulis Siu-Ming Yiu
Department of Computer Science and Information Systems
University of Hong Kong, Pokfulam Road, Hong Kong

ABSTRACT

Using RDBMS to store XML documents is an established trend. However, it fragments the documents into tables and usually requires many joins to answer a query. This may seriously degrade query performance. It was observed that clustering XML documents intelligently based on their structures will alleviate the problem. This is because it only needs to access the cluster(s) of documents that satisfy the query rather than access the whole database. To achieve a good clustering, we propose a computationally inexpensive distance metric. It can be integrated with many available clustering algorithms. Our experiment with real data such as the DBLP database shows that our approach can improve query performance significantly.

Keywords: XML, Clustering, Query.

1. INTRODUCTION

With XML becoming a standard for information exchange, there is an increasing amount of information in XML format that needs to be queried and analyzed efficiently. Besides flat file storage, object-oriented databases and native XML databases, developers have been engaged to use the more mature relational database technology to manage XML data. As a result, a number of specialized techniques have been developed, for processing queries that conform to XML languages, such as XPath and XQuery.

A common strategy of using RDBMS to store XML documents [2][3] [13] is to decompose the structural part of the documents into relational tables that capture the relative position of the various *elements*. Queries are processed by joining these tables, in order to bring back structural (e.g., parent/child) relationships. As this approach decomposes documents and inserts the data into a set of tables, it nevertheless introduces excessive *fragmentation*. This creates a serious negative impact in query evaluation: the number of joins required to process a path expression is generally unacceptably large.

One observation follows this issue : if the collection consists of XML documents with different structures, then the fragmentation problem can be alleviated by *clustering* the documents according to their structural characteristics and storing each cluster in a different set of tables. As a consequence, a query only needs to access the cluster(s) of documents that may satisfy it rather than the whole database.

XML documents have diverse types of structural information (apart from edges) in different refinement levels, e.g., attribute/element labels, edges, paths, twigs, etc. When defining the distance between two documents, choosing a simple structural component (e.g., label, edge) as a basis would make clustering fast. On the other hand, a metric based on highly refined components could make it less efficient and hence impractical. We have observed that using directed edges to define a distance between two XML documents is a good choice. More importantly, this metric can be applied not only on documents but also between sets of documents. Our contributions verified by experiments on real data can be summarized as follows:
1. We show that, if a collection of XML documents consists of documents of different structures, a proper clustering would alleviate the fragmentation problem.
2. We propose a computationally inexpensive distance metric to be defined between documents and between sets of documents, which can be easily embedded in many available clustering algorithms.

The rest of the paper is organized as follows. Section 1.1 discusses related work. Section 2 motivates the study and Section 3 describes the proposed distance metric and a case in which the metric is integrated with a representative clustering algorithms. Section 4 shows how to use MDL to measure the quality of clusters. In Section 5, the applicability of the proposed methodology is evaulated on real XML document collections. Finally, Section 6 concludes the paper.

1.1 Related Work

XML data can be stored in a file system [1], an object-oriented database [7], a relational database [2], or a native XML database system [10]. Using a file system is a straightforward option, which however does not support efficient query processing. Object-oriented database systems allow a flexible storage system of XML files and support complicated query processing. Native XML databases try to exploit features of semi-structured data model in storing XML files. Nevertheless, both object-oriented and native XML database systems are neither mature nor efficient enough for industry adoption. On the other hand, even though relational database technology is not well-tuned for semi-structured data, it is regarded as a practical approach because of its wide deployment in the commercial world.

Many XML database applications decompose the data

and store them in relational tables that capture the structural relationships. A widely used solution is to encode the relative positions of the nodes in the documents using a preorder traverse and store the order in tables arranged under the node labels [2] [3]. Thus, a parent/child queries a/b (or ancestor/descendant queries $a//b$) can be evaluated by first selecting the nodes labelled a and b into two lists and then traverse the two lists in a sort-merge join fashion verifying the structural relationships between the elements. A complex query containing path or twig expressions can be evaluated by decomposing it into a set of binary joins and then merging the partial results. Obviously, fragmentation is a common problem which greatly affect the query performance in these approaches. Our method is to use clustering to alleviate the fragmentation problem so as to answer queries more efficiently.

Clustering is a well studied subject. Recently, Nierman and Jagadish [11] proposed a method to cluster XML documents using the "edit distance" between tree structures. However, computing the edit distance between two documents has a complexity of $O(|A| \cdot |B|)$, where $|A|$ and $|B|$ are their respective sizes [11]. The high computation cost makes this approach unpractical for large data set. In our approach, we cluster graph summaries on directed edges which are much smaller than the original documents and we define a distance metric which is inexpensive to compute. Furthermore, an XML document can be an arbitrary graph rather than a tree because of the explicit element references. For example, both id/idref attribute and XLink construct can create a cross-elements reference). Our methodology can be applied to arbitrary XML graphs, not only trees.

2. MOTIVATION

In order to store XML documents with relational databases, XML documents need to be flattened and fragmented before they are stored in tables. There are several methods for mapping XML documents to relational tables. Each one has a different technique for rewriting semi-structural queries to SQL. To simplify our discussion, we use the mapping and rewriting method in [2], where two tables $ELEMENTS$ and $TEXTS$ are designed to store XML documents.[1]. $ELEMENTS$ table is used to record all the information of elements, including their document numbers, begin and end positions, and levels in the document tree. $TEXTS$ table is used to record all text values, including their document numbers, positions, and levels in the document tree.

In Figure 1, there are four XML documents conform to the following DTD:

$$<!ELEMENT\ A\ (B, C)^* >$$
$$<!ELEMENT\ D\ (B, C, E)^* >$$

Suppose the documents in Figure 1 are partitioned into two clusters such that $Cluster_1$ includes $doc1$ and $doc2$, while $Cluster_2$ includes $doc3$ and $doc4$. The Clustered Schema contains four tables: $ELEMENTS_1$, $TEXTS_1$, $ELEMENTS_2$ and $TEXTS_2$, where $ELEMENTS_1$ and $TEXTS_1$ are used for documents in $Cluster_1$; and $ELEMENTS_2$ and $TEXTS_2$ are used for documents in $Cluster_2$. Obviously, the Original Schema on these four documents contains two tables $ELEMENTS$ and $TEXTS$, where $ELEMENTS$ is the union of $ELEMENTS_1$ and $ELEMENTS_2$; $TEXTS$ is the union of $TEXTS_1$ and $TEXTS_2$.

[1]Since the problem is on clustering XML documents, the choice of mapping and rewriting method does not affect the generality of our result.

doc1: <A><B>bv1</B><C>cv1</C></A>
doc2: <A><B>bv2</B><C>cv2</C><C>cv3</C></A>
doc3: <D><B>bv3</B><C>cv4</C><E>ev1</E></D>
doc4: <D><B>bv4</B><C>cv5</C><C>cv6</C><E>ev2</E></D>

Figure 1: Documents

Suppose a queries $/A/C[contains(., "cv1")]$ is submitted to both the Original Schema and Clustered Schema. The query is to find out all the elements C, which are the children of A and their values contain the string "cv1".

In the Original Schema, we need a self join on $ELEMENTS$ and a join between $ELEMENTS$ and $TEXTS$. In the Clustered Schema, we need a self join on $ELEMENTS_1$ and a join between $ELEMENTS_1$ and $TEXTS_1$. It is clear that the joins between (1) $ELEMENTS_1$ and $ELEMENTS_2$; (2) $ELEMENTS_1$ and $TEXTS_2$; (3) $ELEMENTS_2$ and $TEXTS_1$ are no longer needed. Hence, the cost of processing the query on the Clustered Schema is much smaller than on the Original Schema. This illustrates the advantage of partitioning the documents according to their structure in query processing.

3. CLUSTERING OF XML DOCUMENTS

In this section, we first define a simple and efficient distance metric based on directed edges of XML documents and then show how it can be integrated with an available algorithm to perform clustering on XML documents.

3.1 Distance between XML Documents

As conventional clustering techniques do not have special emphasis on semi-structured data, what would be a proper approach for clustering semi-structured data? It is straightforward to treat the elements of a document as attributes and convert the document into a transaction of binary attributes. Jaccard Coefficient or Cosine function, among various other distance measures, can be used to measure the distance between documents. However, many structurally different documents have almost the same set of elements, for example A/B is entirely different from B/A although they have the same elements.

Since XML documents can often be modelled as node-labelled trees, another option would be to use *tree distance* [15] to measure their distances. In [11], besides node relabelling, insertion and deletion, the tree distance method is refined to allow insertion and deletion of sub-trees, which

makes it feasible to calculate the distance of document trees. However, the cost of computing the tree distance between two documents is expensive (quadratic to their sizes), rendering it unsuitable for large collection of documents. In the following, we introduce the concept of **e-graph** which provides a simple encoding of the edge information in an XML document for the purpose of introducing an efficient distance metric.

Definition 1 *Given a set of XML documents C, the* **edge graph** *(or* **e-graph***) of C, $eg(C) = (N, E)$, is a directed graph such that N is the set of all the elements and attributes in the documents in C and $(a, b) \in E$ if and only if a is a parent element of element b or b is an attribute of element a in some document in C.*

Note that, in the same manner, a path expression q can also be viewed as a graph (N, E), where N is the set of elements or attributes in q and E is the set of element-subelement or element-attribute relationships in q. Given a path expression q which has answer in an XML document X, the directed graph representing q is always a subgraph in the e-graph of X. For simplicity, we will denote the graph of a path expression q also by q.

Theorem 1 *Given a set of XML documents C, if a path expression q has answer in some document in C, then q is a subgraph of $eg(C)$. Also, $eg(C)$ is the minimal graph that has this property.*

The minimality property of $eg(C)$ can be derived from the observation that any proper subgraph of $eg(C)$ will not contain all path expressions that can be answered by any document in C. Thus the e-graph of C is a "compact" representation of the documents in C with respect to the path expressions. Note that the construction of $eg(C)$ can be done efficiently by a single scan of the documents in C.

Corollary 1 *Given two sets of XML documents C_1 and C_2, if a path expression q has answer in a document of C_1 and a document of C_2, then q is a subgraph of both $eg(C_1)$ and $eg(C_2)$.*

It follows from Corollary 1 that if the edge graphs of two sets of documents have few overlapping edges, then there are very few path expressions that can be answered by both of them. Hence, it is reasonable to store them in separate sets of tables. The following distance metric is derived from this observation.

Definition 2 *For two XML documents C_1 and C_2, the distance between them is defined by $dist(C_1, C_2) = 1 - \frac{|eg(C_1) \cap eg(C_2)|}{max\{|eg(C_1)|, |eg(C_2)|\}}$, where $|eg(C_i)|$ is the number of edges in $eg(C_i)$, $i = 1, 2$, and $eg(C_1) \cap eg(C_2)$ is the set of common edges of $eg(C_1)$ and $eg(C_2)$.*

It is straightforward to show that $dist(C_1, C_2)$ is a metric [4]. If the number of common element-subelement relationships between C_1 and C_2 is large, the distance between their e-graphs will be small, and vice versa. It is important to point out here that using e-graphs allows the application of the same metric on documents as well as sets of documents, a property that simplifies the clustering process.

3.2 A Framework for Clustering XML Documents

Employing the distance metric in Definition 2, we perform XML clustering in two steps:

Step 1. Extract and encode edge information in e-graphs: this step scans the documents, computes their e-graphs and encodes them in a data structure.

Step 2. Perform clustering on the encoded e-graphs: this step applies a suitable clustering algorithm on the data structue to generate the clusters.

Initially, the e-graphs of all the documents are computed and stored in a structure called EG. An e-graph can be represented by a bit string which encodes the edges in the graph. Each entry in EG has two information fields: (a) a bit string representing the edges of an e-graph and (b) a set containing the ids of all the documents whose e-graphs are represented by the bit string.

In general many documents may share the same e-graph, therefore the size of EG may be much smaller than the total number of documents. In the extreme case where the size of EG is very large, general approach such as sampling can be used.

Once EG is computed, clustering is performed on the bit strings. Therefore, we transform the problem of clustering XML documents into clustering a smaller set of bit strings, which is efficient and scalable.

In our framework, we have separated the encoding and extraction of the structural information from the clustering part. So that any appropriate algorithm could be used in the clustering of the e-graphs. In the following, we will explain how we apply a representative clustering algorithm DBSCAN [6], on the e-graphs.

3.3 Integration with DBSCAN

DBSCAN [6] is a density based clustering algorithm. There are two global parameters: Eps, which is the radius of a neighborhood; $MinPts$, which is the minimum number of points that a neighborhood should have in order that the point under consideration, i.e., the center of the neighborhood is considered as a Core Object.

$NEps(p)$ stands for the set of neighbors (objects) of p inside its neighborhood defined by Eps. A point p is **directly density-reachable** from a point q if (1) p belongs to $NEps(q)$ and (2) $|NEps(q)| >= MinPts$.

A point p is **density-reachable** from a point q if there is a chain of points $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that $p_i + 1$ is directly density-reachable from p_i for all i.

In DBSCAN, there are three possible labels for a point: unclassified, classified and noise. If a point is labelled as unclassified, it will be used to generate possible cluster. If a point is labelled as classified, then it already belongs to a cluster and need not be processed further. If a point is labelled as noise, itself can not be used to generate possible cluster; however it can be swallowed by a cluster generated from an unclassified point later in the processing.

The pseudocode of the integration with DBSCAN is shown in Figure 2.

The first two lines extract edge graphs from document

```
/*Input  X: a set of XML documents*/
/*Input  Eps: radius of a neighborhood */
/*Input  MinPts: minimum number of points in a dense neighborhood */
/*Output  C: a set of clusters;*/
1)  EG=computing_EG(X);
2)  DIST=compute_distance(EG);
3)  neighbors=compute_neighbors(DIST, EG, Eps);
4)  for i = 1 to |Eg| {
5)    if (cluster-status(EG_i)==unclassifed)
6)      c_z=get-all-density-reachable-points(EG_i,Eps,MinPts,neighbors);
7)      if (corepoint(EG_i)) //return true if EG_i is a core point
8)        C=C ∪ c_z
9)        change-cluster-status(c_z, classified)
10)     else
11)       change-cluster-status(EG_i, noise)
12)  }
```

Figure 2: Integration with DBSCAN

set and compute the distance between each pair of edge graphs. Then *compute_neighbors* (line 3) is called to maintain a list of neighbors for each e-graph. The loop (lines 4–12) performs the clustering. It works as follows: 1) arbitrarily select a point (e-graph) EG_i which is unclassified (line 5); 2) retrieve all points that are density-reachable from EG_i (line 6); 3) if EG_i is a core point, then all the retrieved points forming a cluster (line 8) and are labelled as classified (line 9); 4) otherwise, no points are density-reachable from EG_i and it is labelled as noise, then go to line 4 to process the next point.

Complexity: Suppose m is the number of distinct e-graphs in EG. In Figure 2, we need to find out all the neighbors for each XML documents and this can be done in $O(m^2)$. During the clustering, the time used is to traverse all points and check their neighbors. Therefore the total time cost is bounded by $O(m^2)$. Since we need to store the distances of all pairs of e-graphs in line 2, and maintain the neighbors for each point, the total space cost is about $O(m^2)$.

3.4 Query Rewriting

Most methods for managing XML data in relational tables provide some query rewriting mechanisms to transform a semi-structured query like XQuery to SQL. In our approach, data is partitioned into clusters. Therefore we need to identify which cluster(s) would likely have answers for a given query before using available rewriting mechanism. This is straightforward, given a query, if the e-graph of the documents in the cluster contains the query, the associated cluster may have answers for the query and hence should be evaluated against.

4. EVALUATIING THE QUALITY OF CLUSTERS WITH MDL

Different clustering algorithms, or even the same clustering algorithm(with different parameters) can produce clusters of different types. We need some measurements to evaluate the quality of clusters. Here we apply the **Minimum Description Length (MDL) Principle** [12] to quantitatively measure the different clustering results. MDL was originally designed to determine good data models. The basic idea is: for each model, the total number of bits required to encode both the data, and the model itself is calculated; the model requiring the least number of bits is considered to be the best.

Given a set of clusters found by an algorithm, a DTD is derived [8] for each cluster such that all documents in the cluster satisfy it. Then we sum the total number of bits required to encode both this DTD, and all the documents in the corresponding cluster. Finally, the total number of bits required for each cluster is aggregated to get the total cost of the given set of clusters. The lower the total cost, the higher the quality of the clusters.

The encoding method we use is provided by [8]. Here we show the method with a simple example, (please refer to [8] for more detail). Given a DTD as $<!ELEMENT \quad a(b|c|d|e)*) >$, the string "$(b|c|d|e)*$", which contains 8 different characters, has a length of 10 in terms of characters. The number of bits needed to encode this string is 10*log8. Given an expression such as "bbdce", we can encode it in the following way: first, set "*" to be 5, which means that the expression is repeated 5 times. Then each time, we need to indicate which character is chosen, because log4 bits are necessary to encoding "b,c,d,e", the encoded string is "5 00 00 10 01 11", the first two "00" stands for "b", the last "11" stands for "e". Of course, a DTD can be composed by a set of definitions of elements, in this case, we can just sum them to get the total cost.

5. QUERY ENHANCEMENT ON REAL DATA

In this section, we investigate whether a Clustered Schema brings in better query performance than the Original Schema. The data set we use is the XML DBLP records database [5], which contains about 300,000 XML documents composed by 36 elements. All documents contain elements such as *author*, *title* and *year*. Overlap among documents' elements is a common scenario. Experiments are carried out in a computer with 4 Intel Pentium 3 Xeon 700MHZ processors and 4G memory running Solaris 8 Intel Edition. We limit the memory usage to 100M and use only one processor.

We defined five types of queries based on the structure of existing documents. The five query classes are (the first three are written in XPath format):

Q1 : $/A_1/A_2/\cdots/A_k$; all possible absolute XPaths in the documents.
Q2 : $/A_1/A_2/\cdots/A_k[text()=$"$value$"$]$; same as Q1 except that an additional condition is added to make the text value of the last element equals to "value", which is a string randomly selected from the data.

Q3 : $/A_1/A_2/\cdots/A_k[contains(.,\text{“substring”})]$; same as Q1 except that the additional condition is to make a randomly picked “substring” contained in the text value of the last element.

Q4 : find the titles of articles published in the VLDB Journal in 1999.

Q5 : find the names of authors which have at least one journal article and one conference paper.

Because path expressions are the basic units in composing XML queries, we use the first three queries to test the performance of processing path expressions. Q4 and Q5 are defined to test the joins among path expressions. The RDBMS we used is the Oracle 8i Enterprise Edition release 8.1.5. All the above five queries are translated to SQL and executed on the RDBMS. We also use the mapping technique in [13] and the schema is denoted by SM1. We use the mapping technique in [2] and the schema derived is denoted by SM2. DBSCAN are used to generate the clusters and the Clustered Schema are built based on the found clusters.

5.1 Query Improvement on the Clustered Schema by Using DBSCAN

To find a proper value of input parameters, we have done experiments with different setting of Eps and $MinPts$, where Eps=0.2, 0.25, 0.3, 0.35, 0.4 and $MinPts$=100, 200, 500. In all the results, we find that when Eps=0.2 and $MinPts$=200, the six resulting clusters lead to a minimal encoding cost using MDL. Therefore they are used to generate the Clustered Schema. Figures 3 shows the query performance speed-up when the Original Schema is compared with the Clustered Schema.

Figure 3: Speed up Ratios for Queries

Each distinct path expression conforming to Q1, Q2, and Q3 in the documents is submitted as a query to the Original Schema and the Clustered Schema. The speed-up ratios for each query type are averaged, the improvement on path expressions is quite substantial. We should mention here that the speed-up ratios of the distinct path expressions in Q1 in fact ranges from 1.4 to 44, because some paths may need to join more tables than the others. The improvement for Q4 and Q5 in Figure 3 is smaller. Comparing the queries in Q4 and Q5 with Q1, it can be seen that the path expressions in Q4 and Q5 involve less joins than those of Q1. This explains why the improvements for Q4 and Q5 are less impressive.

It is clear the query processing cost drops dramatically in Clustered Schemas, since many unnecessary joins between irrelevant parts of the original tables are avoided.

5.2 More Experiments on Real Data

We have also carried out the same experiments of Q1 to Q5 on StackTree [2] and PathStack [3], Figure 4 show the results. It is clear the improvements is similar to what we have seen in Figure 3, which again demonstrates that clustering is a good choice to improve query performance.

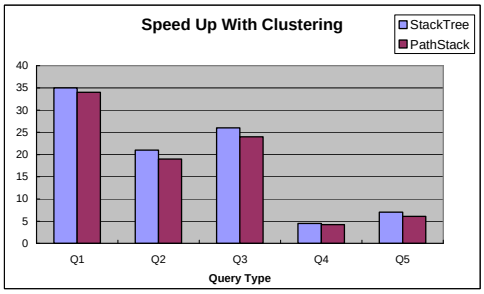

Figure 4: Speed up Ratios for Queries

5.3 Comparison with Algorithm Based on Tree-distance

As well as studying the performance of DBSCAN, we also compared it with the clustering algorithm ESSX, proposed in [11]. ESSX hierarchically merges clusters of documents using the tree-edit distance. At each step of the algorithm, the pair of clusters with the smallest average distance between their documents is merged. The edit distance between two trees is defined as the minimum cost required to transform one tree into the other. This cost is computed by summing the cost of the primitive operations involved in the transformation (i.e., node insertion, node deletion, node renaming, subtree insertion, and subtree deletion). Since the cost of computing tree distance on XML documents is very high, we only ran ESSX on a sample of 40 documents randomly selected from the DBLP database. [2]

In the 40 documents, there is a natural partitioning: 10 documents belong to *proceedings*; 10 to *phdthesis*; 10 to *journals*; 6 to *books*; and 4 to *incollections*. By setting the number of clusters k to 5, DBSCAN discovered five clusters that match exactly the original partitioning. When running ESSX, the cluster number was set to 5, nd the cost of node relabelling, insertion and deletion were all set to 1, and the cost of subtree insertion and deletion, ranged from 0 to 10. Interestingly, the clustering results were the same for all the values in this range. However, the clusters generated were very different from the original partitioning. One of the five clusters contained 30 documents from *proceedings*, *phdthesis*, *books* and *incollections*. The remaining 10 *journal* documents were distributed into 4 clusters, A, B, C and D, with

[2] 40 documents is already larger than the dataset used in [11], which contains only 20.

$|A| = |B| = |C| = 1$, and $|D| = 7$. We found that none of the documents in D contained the tag *cite*, while documents in A, B, C contained many instances of *cite* as tree leaf. Therefore, it is impossible to use a subtree editing operation to convert a document in D to a document in A, B, or C. Only node insertion or deletion can be used to convert the source tree to the target tree in this case. This explains why the different cost parameters of subtree editing operation has no effect in the clustering. Since node insertion has a positive cost associated with it, the difference in the number of *cite* tags between two documents would affect the edit distance between them. Because of this, the journal documents form 4 clusters, because some of them have many more *cites* than the others.

The time required for ESSX to cluster the 40 documents is 530 seconds, whereas DBSCAN runs in less than 2 seconds, including the I/O cost. This demonstrates that DBSACN is not only more effective, but also more efficient than ESSX at performing clustering.

After producing the DTDs on the clusters discovered by DBSACN and ESSX, we use MDL to measure the encoding costs and notice that: the cost of encoding DBSACN's DTD is 302 bits less than that of ESSX. Similarly, to encode the documents, ESSX requires an additional 60 bits. MDL shows that the quality of clusters given by DBSACN is good.

6. CONCLUSION

Excessive fragmentation of XML documents into relational tables may greatly degrade the query performance. Our observation is: if the collection consists of XML documents with different structures, then the fragmentation problem can be alleviated by *clustering* the documents according to their structural characteristics and storing each cluster in a different set of tables. Based on this observation, we proposed a simple and effective distance metric which can be easily integrated into many available clustering algorithms.

Our goal is not to design new clustering algorithm or advocated any new schema design method for storing XML documents. However, an inexpensive and effective distance metric is a necessary factor if a clustering is to be performed on collections of XML documents to produce useful partitions. Also, we have demonstrated that it is beneficial to identify clusters by structure and they can improve the performance of the query processing.

References

[1] S. Abiteboul, S. Cluet, T. Milo, "Querying and Updating the File," **Proc. 19th Int'l Conf. Very Large Data Bases**, pp. 73–84, Morgan Kaufmann, 1993.

[2] S. Al-Khalifa, HV Jagadish, N. Koudas, JM Patel, D. Srivastava, and Y. Wu. "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," **Proceedings of International Conference on Data Engineering**. 2002.

[3] N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. **Proc. of ACM SIGMOD Conf.** 2002.

[4] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," **Pattern Recognition Letters**, vol. 19, no. 3, pp. 255–259, 1998.

[5] DBLP XML records, **http://www.acm.org/sigmod/dblp/db/index.html**, February 2001.

[6] M. Ester, H. Kriegel, J. Sander and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," **Proc. Second Int'l Conf. Knowledge Discovery and Data Mining**, pp. 226–231, 1996.

[7] Excelon, **http://www.odi.com/excelon**.

[8] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim XTRACT: A System for Extracting Document Type Descriptors from XML Documents. **Proc. of ACM SIGMOD Conf.** Pages 165–176. USA. 2000.

[9] R. Kaushik P. Shenoy P. Bohannon and E. Gudes, "Exploiting Local Similarity for Indexing Paths in Graph-Structured Data," **Proc. 18th Int'l Conf. Data Engineering**, 2002.

[10] J. McHugh, S. Abiteboul, R. Goldman, D. Quass and J. Widom, "Lore: A Database Management System for Semistructured Data," **SIGMOD Record**, vol. 26(3), pp. 54–66, September 1997.

[11] A. Nierman and H.V. Jagadish, "Evaluating Structural Similarity in XML Documents," **Proc. Fifth Int'l Workshop on the Web and Databases**, June 2002.

[12] J.Rissanen. Modeling by Shortest Data Description. **Automatica**. Vol 14, Pages 465–471. 1978.

[13] T. Shimura, M. Yoshikawa, S. Uemura, "Storage and Retrieval of XML Documents using Object-Relational Databases," **Proc. 10th Int'l Conf. Database and Expert Systems Applications**, pp. 206–217, Springer-Verlag, 1999.

[14] O. Zamir, O. Etzioni, O. Madani and R. M. Karp, "Fast and Intuitive Clustering of Web Documents," **Proc. Second Int'l Conf. Knowledge Discovery and Data Mining**, pp. 287–290, 1997.

[15] K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance between Trees and Related Problems," **SIAM Journal of Computing**, vol. 18(6) pp. 1245–1262, 1989.