XML Compression Improvements Based on the Clustering of Elements

Pavel Hruška, Jan Martinovič, Jiří Dvorský, Václav Snášel
Department of Computer Science,
VŠB - Technical University of Ostrava
17. listopadu 15, 708 33
Ostrava-Poruba, Czech Republic
{pavel.hruska,jan.martinovic,jiri.dvorsky,vaclav.snasel}@vsb.cz

***Abstract*—This paper focuses on improvement of compression of XML documents based on clustering and rearranging of XML elements within XML documents. Such transformed XML documents can be efficiently compressed.**

I. Introduction

The modern information society produces immense quantities of textual information. Storing text effectively and searching necessary information in stored texts are the tasks for *Information Retrieval Systems* (IRS). The size of an IRS increases with the increasing size of available external memories of computers. Therefore, it is now possible to have a several gigabyte IRS on one DVD. Similarly, with the growth of Internet it is possible to have an easy remote access to an extensive IRS, which is stored in an even bigger disk array that operates on an Web server. We can only expect even faster growth of memory capacity requirements in future. The information explosion can be avoided basically in two ways:

1) Extensively - by purchasing higher capacity memories, or
2) Intensively - by storing data in memories in a better way.

The first solution is not interesting in terms of research. The key to the second solution is *data compression*. The database of a typical IRS is a *textual database*, which stores all information that is necessary for the function of the IRS. Textual databases typically consist of the three following parts:

- Document full-texts that form a document collection
- Data structures for searching documents
- List of document identifiers and of their attributes and other auxiliary structures

Haskin claims in [11] that the size of textual database auxiliary structures (i.e. except actual document texts) makes up 50% to 300% of the size of original documents. This implies that a textual database is a suitable material for compression. You only have to use one of lossless compression methods to save more or less space.

However, the problem of compression in IRS is not as simple as it seems at first sight. On the one hand, compression saves space for data, however, on the other hand, it may entail a certain operation overhead i.e. adding certain amount of time to the cost of accessing the data. Also, the space saving must be significant to be useful. Therefore, the objective is not to compress the textual database as a whole. This usually does not lead to good results since individual parts of an IRS contain redundancies of different types; different data structure types are based on a different model, according to which it is possible to determine the best compression method.

Experiences show that it is useful to consider, analyze and design the best compression method when storing extensive textual databases. It also proves to be desirable to study highly specialized compression methods that are convenient only for a certain data type in an IRS. Even saving e.g. one bit in data structures for searching and the improvement of text compression ratio in an IRS by one percent result in savings of tens of megabytes.

II. XML

One frequently used method of storing information in IRS is XML language. The XML is now a very popular language which is used in many areas of exchange and information storage. Its advantage is particularly versatility, allowing everyone to form his/her own language for various applications. XML stores all information in text form – the data itself, as well as information about the document structure. The structure of the document consists of tags and attributes that give the meaning of content contained therein. Therefore, XML is often called as self-described data [23], [15].

The main disadvantage of XML language is it's verbosity – all information is stored in textual form (unicode characters), and semantic information (tags) is repeated in each instance of particular element. Repeated tags as well as XML white-spaces used as XML formating and textual representation of all data increases data storage needs and the final XML document is usually much larger in comparision to native binary formats designed for specific purposes. Another important fact we must consider is that XML document must be parsed before we can access it's data and/or structure. Parsing XML utilizes CPU and memory resources and parsing some large XML document may be critical for systems with such limited resources.

III. XML compression

In order to reduce space demands of XML document, some kind of compression should be used. In general, XML compression is looseless. There are available many XML compressors up to date and we can classify them with respect of two characteristics. The first classification is based on the opportunity to work with XML in compressed form – on their ability of supporting queries.

Queriable XML Compressors: These compressors focuses on compressing XML document and also preserves oportunity to query such compressed XML format. These methods compress the XML document in pieces, their efficiency is usually the worse (in comparison with the compression of XML as a whole). XML compressors such as XGrind, XQZip and XPRESS belong to this category.

Non-Queriable XML Compressors: The XML compression without query support is our area of research interest. This group of compressors compresses XML without support of any further XML operations to be processed over the compresed version of XML document. Main purpose of those compressors is to achieve better compression ratios. Such compressed XML documents are supposed for example to be archived or transfered over networks (internet, intranet, slow WAN-links, ...). If some edit operations are required, these methods require decompression of XML as a whole, and storage of such modified document results in recompression of the whole XML document. Compared to the previous group of compression methods they are characterized by greater compression efficiency. The following text will only deal with these methods.

Second classification of XML compressors is based on their awareness of the structure of XML documents.

General Text Compressors: Group of compressor utilizing standard general purpose text compression techniques – this is logical approach as we can consider XML document as plain text document, which consists of a series of symbols. Compressing XML document as text is straightforward as we can use standard compression algorithms. Using such compressors has some advantages – simplicity and speed of deployment, because text compression algorithms and tools are commonly available. Deflate or BZip2 represents contemporary industrial standards. The prospective compression method is based on PPM algorithm [6]. The PPM algorithm provides very good results especially for compression of documents written in some natural language.

XML-aware compressors: These compressors are aware of XML structure and are designed to achieve better compression ratio using semantic information present in the XML structure. This approach exploits the tags in XML document and assumes that the tags give specific meaning to content inside the tags. The common feature of this approach is the separation of data structures from content. Another common feature is that XML-aware compressor preprocesses XML data (using semantic information) and final compression is done by conventional general purpose text compressor (such as Deflate or BZip2). The traditional compression method, such as Deflate, can compress the transformed data, XML document, with even higher efficiency then original data. Designing XML-aware compressor consits of two parts – first: deeply understand back-end compressor techniques; second: data preprocessing design according to back-end compression properties to achieve better compression. XMill is the most known representative of such approach.

IV. Queriable XML compressors

Altough this group of XML compressors is not area of our research, it is very interesting to see it's evolution. It is important to have in mind that compression ratio is not the only one parameter we should compare among compressors. Supported range of operations, including e.g. range of XPath implementation and/or query performance is quite important parameter of such queriable XML compressor.

In [1] Tolani and Haritsa have presented the first XML-aware compressor supporting querying without the need to full decompres XML document. XGrind uses homomorphic compression. Homomoprhic compression does not separate structure and data, it preserves original structure – tags and data are simply replaced by their encoded values. Element and attribute names are encoded using a dictionary-based encoding, and character data is compressed using semi-adaptive Huffman coding [12]. Semi-adaptive encoding rules are independent to the locations of the data (statistical information is gathered in prelimminary scan of the file) and this is very important for the ability to decompress only a choosen part of compressed XML without need to know any preceeding symbols. Such homomoprhic compressed document can be processed by standard XML-tools (view XML, index XML, ...). XGrind has some limitations. It can only handle exact-match and prefix-match queries on compressed values and partial-match and range queries on decompressed values [1]. However, several operations are not supported by XGrind.

In [19] Min et al. have described another homomorphic queriable XML compressor, XPRESS. It uses a reverse arithmetic encoding method to encode the labels and paths of XML documents. To improve compression, XPRESS uses proper encoders for data values based on their automatic inferred type information. Compression is semi-adaptive.

In [5] Wilfred and Ng have descibed non-homomorphic compressor XQZip. XQzip addresses both the compression and query performance problems of previously described XML compressors. It introduces an indexing structure called the Structure Index Tree (or SIT) that removes the duplicate structures in an XML document to improve query performance. XQzip avoids full decompression by compressing the data into a sequence of blocks which can be decompressed individually and at the same. XQzip supports a wide scope of XPath queries such as multiple, deeply nested predicates and aggregation – supports a much more expressive query language than its counterpart technologies such as XGrind and XPRESS.

In [3] Al-Hamadani, Alwan and Lu proposed a another XML compression technique that obeys the structure of the XML documents (it is homomorphic compression) and provides the ability to querying the compressed document with both content and structure (CAS) queries type. XML elements and attributes names are encoded by using fixed-point dictionary-based technique. Other XML data are organized into special containers according to their type and path from the root attribute, and the containers are compressed using the same fixed-point technique. Based on type, each container is compressed using different encoding techique (such as integer, floating-point or enumerates).

V. Non-queriable XML compressors

There are two groups of non-queriable XMLcompressors – schema independent and schema dependent [23]. In schema depented compressors both of the encoder and decoder must have access to the document schema information to achieve the compression process. Although schema dependent compressors may be able to achieve better compression ratios (in theory), in practice they are not very popular and widely used because of schema might not always be available for specific XMLs and thus for this specific XML schema dependant XML compressor cannot be used. [23] lists and shortly describes some schema dependant XML compresors such as DTDPPM, XAUST and others.

In [16] Liefke and Suciu have described implementation of an XML-aware compressor tool. XMill compressor uses zlib (gzip function library), a set of specific compressors for simple data types and, possibly, user defined compressors for application specific data types. XMill applies three principles to compress XML data: 1) Separate structure from data, 2) group releated data items and 3) apply semantic compressors. Those ideas, introduced by XMill, were followed by many other XML-aware compressors. The most imporatant part of compression is data grouping based on their relative path in the XML structure tree. XMill assumes that tags gives semantic meaning to data enclosed within them and that relevant data grouped together will be compressed much more effectively by general text compressor. Data are grouped into so-called containers and each container is separately compressed by back-end compressor, which may vary depending on container type. Even XML structure, encoded using dictionary-based coding, is stored in separate container and thus also compressed. Back-end compressors utilized by XMill are GZip [GZIP], BZip2 [BZIP2] and PPM [6].

In [15], Li describes slightly modified XMill algorithm called XComp. Some experiments were performed with Huffman and GZip compression and the result was only a small improvement over the original XMill compression.

In citeXMLPPM, Cheney has presented XMLPPM as XML compressor which uses a Multiplexed Hierarchical PPM Model called MHMPPM. XMLPPM is based on parsing XML into stream of encoded SAX events (ESAX, Encoded SAX) and compressing it using compressors such as gzip or ppm. Cheney states that using gzip or ppm, ESAX is only 7% or 1% worse than XMill, respectively. Using PPM, the ESAX is encoded by one of four multiplexed PPM models based on its syntactic structure (elements, characters, attributes, and others).

In [2] Adiego, Navarro end Fiente described a compression model for semistructured documents called Structural Contexts Model (SCM). Authors has presented various variants SCM, such as SCMHuff (which utilizes Huffman compression) and SCMPPM (utilizing PPM compression). The idea behind SCMPPM is to use a separate model to compress the text that lies inside each different structure type – that means in every different XML tag. Ideas behind SCMPPM and previously described XMLPPM are very close. SCMPPM useses a bigger set of PPM models than XMLPPM because it uses a different model for each different tag name.

In [24] Toman proposed a novel syntactical XML compression scheme which makes use of probabilistic modeling of XML structure. It is based on syntactical compression techniques of [20] algorithm by Nevill-Manning and Witten [Sequitur], and grammar-based codes proposed recently by Kieffer and Yang [Kieffer]. It is utilizing a fact, that XML document could be described using context-free grammar. Experimental results listed in [Exalt] are shown to be comparable to XMill results in terms of compression ratio.

VI. Cluster Analysis

Cluster analysis is the process of separating documents, with the same or similar properties, into groups that are created based on specific issues. We will call these groups of documents *clusters* [13]. Clustering may be applied to terms or documents when working with documents in IR systems. Term clustering can be used for creating a thesaurus. Joining similar documents to a cluster may be done by increasing the speed level for searching in search engines. The reason for carrying out a cluster partitioning is explained in *hypothesis about clusters* [14]:

> *When documents are in close proximity, they are relevant to the same information.*

We are going to focus on clustering documents and our work can be summarized by the following two steps: creating a cluster and searching for relevant clusters [10].

The process within which the ideal cluster partitioning for sets of document is searched, and within which there are mutually similar documents, is called *clustering*. The cluster is then formed mutually by a set with similar documents.

In an ideal situation, the clustering procedure should accomplish two goals: correctness and effectiveness [10]. The criteria for correctness follow:

- methods should remain stable while collections grow or, in other words, distribution into clusters should not drastically change the addition of new documents,
- small errors in document descriptions should be carried over as small changes in cluster distributions into clusters,
- a method should not be dependent on its initial document ordering.

VII. Topical Development

In the paper [8] we defined ϵ-k-ball and its behavior in a space that does not maintain the rules of triangle inequality. Now, we define the concept k-path, for which the term ”topical development” will be used.

The definition of k-path: for the given $x \in \mathbb{X}$ and $k \in \mathbb{N}^+$, the set $B^k(x) = \{y \in \mathbb{X}; x_1, \ldots, x_k \in \mathbb{X}, x = x_1, y = x_k\}$ is called the k-path centered at the point x.

We can present topical development as a path leading away from the initial document, through similar documents and towards other documents pertaining to this document.

We can illustrate this path in a vector space, where our document forms nodes. The edges between these nodes evaluate their similarity. If this path satisfies the conditions for k-path we can say that it is a proper representation of topical development.

Thematic similarity between documents in text collections is influenced by terms that occur in the document. Let us take a document, which describes a given topic, from a collection of documents. There may be other documents in our collection of documents that either entirely, or partially, shares the same topic (problematic). These documents, however, may use a part of another word to describe the given topic. The difference in this word may be caused by various reasons. The first document may direct a set of words toward the topic and the second document may include a synonym or it may be more focused on other circumstances influenced by the chosen topic (a new fact, a political situation, a new problem trend and so on).

A. Algorithm Acquired of Topical Development

For acquiring topical development from hierarchal clustering, we will define the algorithm *TOPIC-CA*, which uses the amount of documents in the development as a hindrance.

Definition 7.1: The TOPIC-CA algorithm (see Algorithm VII.1) for acquiring topical development is defined with the aid of a dendrogram D_{Tree} as list $S_T = TOPIC\_CA(d_q)$. Where d_q is a node in the dendrogram for which we want to generate a topical development.

Algorithm VII.1 Algorithm TOPIC-CA – function $TOPIC\_CA$

```
function TOPIC_CA($node \in D_{Tree} \cup null$)
    $L \leftarrow$ Empty list
    if $node \neq null$ then
        AddNodeToEnd($L$, $node$)
        while $node \neq null$ do
            $sibling \leftarrow$ SIBLING($node$)
            $L \leftarrow$ SUB($sibling$, $L$)
            $node \leftarrow$ PARENT($node$)
        end while
    end if
    return $L$
end function
```

The advantage of using this algorithm for acquiring topical development is low time and space requirement during querying. For searching topical development, we need a dendrogram with pre-calculated similarity for each individual node of the dendrogram. The disadvantage is the time required to create the dendrogram. A calculation of the hierarchal cluster is performed during the creation of a textual database, so users entering queries into the IRSs are not influenced by this factor.

Algorithm VII.2 Algorithm TOPIC-CA – function Sub

```
function SUB($node \in D_{Tree} \cup null$, list $L$)
    if $node = null$ then
        return $L$
    end if
    $sibling \leftarrow$ SIBLING($node$)
    if $node \in$ leaf nodes of $D_{Tree}$ then
        AddNodeToEnd($L$, $node$)
    else if $sibling \neq null$ then
        $siblingLeft \leftarrow$ LEFTCHILD($sibling$)
        $siblingRight \leftarrow$ RIGHTCHILD($sibling$)
        $sim_{Left} \leftarrow$ SIM($node$, $siblingLeft$)
        $sim_{Right} \leftarrow$ SIM($node$, $siblingRight$)
        if $Sim_{Right} \leq Sim_{Left}$ then
            $L \leftarrow$ SUB($siblingLeft$, $L$)
            $L \leftarrow$ SUB($siblingRight$, $L$)
        else
            $L \leftarrow$ SUB($siblingRight$, $L$)
            $L \leftarrow$ SUB($siblingLeft$, $L$)
        end if
    end if
    return $L$
end function
```

The following functions are used in the algorithm:

- $TOPIC\_CA$ – main function for calculating topical development (see Algorithm VII.1),
- Sub – function for recursive dendrogram outlet (see Algorithm VII.2),
- Sim – calculation for similarity of a given cluster in a dendrogram D_{Tree} to a neighbor's descendant cluster (see Algorithm VII.3),
- $Sibling$ – acquired neighboring nodes,
- $Parent$ – acquired parent nodes,
- $LeftChild$ – acquired left descendant,
- $RightChild$ – acquired right descendant,
- $AddNodeToEnd$ – addition of a document to resulting topical development. If the calculation of documents in a topic is equal to the required amount of documents, algorithm $TOPIC\_CA$ ends (to simplify the process, it is left out of algorithm $TOPIC\_CA$).

VIII. Using Topical Development for XML Document Compression Improvement

Ordering of elements in input XML document has not yet been taken into consideration within the general description of compression methods. The compression method works properly for any type of elements ordering. Original ordering given by XML document itself is probably the simplest of elements ordering options, i.e. elements are compressed in the same order as they are written in XML document. Seeing that compression methods are based on searching repeated parts of texts, it is easy to surmise that this ordering option is

Algorithm VII.3 Algorithm TOPIC-CA – function Sim. Calculated proximity of cluster n_1 to a descendant of a neighboring cluster n_2 in the hierarchy

```
function SIM($n_1 \in D_{Tree} \cup null$, $n_2 \in D_{Tree} \cup null$)
    if $node_1 = null \vee node_2 = null$ then
        return 0
    end if
    $c_{n_1} \leftarrow$ centroid created from all leafs nodes in $n_1$
    $c_{n_2} \leftarrow$ centroid created from all leafs nodes in $n_2$
    $sim \leftarrow$ similarity between $c_{n_1}$ and $c_{n_2}$
    return $sim$
end function
```

not necessarily the best solution. Improvement of compression performance can be achieved by reordering of XML elements. We improve the ordering of XML elements by moving similar elements nearer to one another.

This improved ordering can be achieved using a cluster analysis. Of course, a cluster analysis is very time consuming so that it is counterproductive to perform the analysis in order to enhance compression performance alone. However, when compression methods for IR system are developed, results from a cluster analysis can be used in query processing [9], [17] and vice versa. Cluster analysis originally performed solely for query processing can be incorporated to compression.

Incorporating a cluster analysis to improve a compression is common in methods that compress inverted indexes (includes a list of documents for every indexed term). These methods, using hierarchical clustering [4] or clustering algorithms, resemble the k-means [21].

However, the question of how to convert a hierarchical tree structure of clusters to a linear list of XMLO elements still remains. The answer is to use topical development [22], [8], [18]. The topical development uses one element that specifies a topic the as starting point of a topic development searching process. This starting element can be chosen arbitrarily, usually the leftmost node in cluster hierarchical tree.

IX. Experimental Results

A. Experimental XML Files

Several XML data files were used in our experiments, see Table I. The XML files *psd7003.xml*, *dblp.xml*, *swissProt.xml*, and *nasa.xml* are free available at http://www.cs.washington.edu/research/xmldatasets/. These files contains different databases stored in XML format[1]. Several thousands files from Wikipedia XML corpus [7], http://www-connex.lip6.fr/~denoyer/wikipediaXML/, were merged to produce test file *wiki.xml*. The Table I provides original sizes of test XML documents and also provides informal time complexity of clustering used in our experiments.

[1]Details can be found at specified URL.

TABLE I
XML Test Files

File	S_0	T_c
psd7003.xml	716,860,101	0:27:08
dblp.xml	714,338,879	0:52:24
wiki.xml	541,873,094	0:38:10
swissProt.xml	114,820,211	0:12:41
nasa.xml	25,054,691	0:00:19

TABLE II
XML Compressed As Text Without Clustering

(a) Size of Compressed XML

File	CS^t_{gz}	CS^t_{bz}	CS^t_{pp}	CS^t_{lz}
psd7003	104,095,774	76,760,863	65,470,863	**61,473,674**
dblp	117,402,883	77,896,607	**56,460,030**	77,717,175
wiki	105,741,871	80,597,419	**59,888,861**	72,825,200
swissProt	13,790,955	8,724,363	**5,914,738**	6,775,538
nasa	3,723,344	2,752,252	**1,945,651**	2,415,560

(b) Compression Ratio

File	CR^t_{gz}	CR^t_{bz}	CR^t_{pp}	CR^t_{lz}
psd7003	14.52	10.71	9.13	**8.58**
dblp	16.44	10.90	**7.90**	10.88
wiki	19.51	14.87	**11.05**	13.44
SwissProt	12.01	7.60	**5.15**	5.90
nasa	14.86	10.98	**7.77**	9.64

B. Experimental Results

Several experiments with transformation of elements inside XML files were preformed. Size of compressed file and compression ratio were observed during the experiments. First XML documents were compressed as plain text. The size of compressed XML documents using given compression methods are provided in Table II(a), compression ratio are given in Table II(b). It is obvious, that the best results are obtained using PPM method with the exception of LZMA method in one case[2]. Table IV shows the compression ratios after the clustering of XML files and also illustrates the potential benefits of transformation of XML documents. As can be seen from the Table IV, the transformation of XML documents has positive impact on compression – δ parameter is in most cases lower than 1.

The second set of experiments was performed with an XML-aware compressors XMill. As in previous set of experiments, first XMill was run without any transformation of input XML documents. Sizes of compressed XML documents and corresponding compression ratios are given in Tables V(a) and V(b). The test XML documents were transformed using clustering in the next step of our experiments. The Table VI shows the same as in the previous case, the improvement or deterioration of the results using this method.

X. Conclusion

The present information society creates huge quantities of textual information. This information explosion is being handled using Information Retrieval Systems. Their tasks are effective storage and searching in the text collections. The

[2]The notation for all tables is given in Table III.

TABLE IV
XML COMPRESSED AS TEXT WITH CLUSTERING

File	CR^{tc}_{gz}	$\delta(CR^{tc}_{gz})$	CR^{tc}_{bz}	$\delta(CR^{tc}_{bz})$	CR^{tc}_{pp}	$\delta(CR^{tc}_{pp})$	CR^{tc}_{lz}	$\delta(CR^{tc}_{lz})$
psd7003	13.64	0.9392	10.51	0.9814	8.95	0.9799	**8.53**	0.9943
dblp	16.52	1.0051	10.79	0.9891	**7.87**	0.9952	10.74	0.9869
wiki	17.94	0.9192	13.06	0.8779	**10.09**	0.9126	12.65	0.9414
SwissProt	10.40	0.8660	6.84	0.8998	**4.94**	0.9590	5.77	0.9771
nasa	14.01	0.9423	10.61	0.9659	**7.75**	0.9980	9.58	0.9931

TABLE VI
XML COMPRESSED WITH CLUSTERING USING XMILL

File	CR^{xc}_{gz}	$\delta(CR^{xc}_{gz})$	CR^{xc}_{bz}	$\delta(CR^{xc}_{bz})$	CR^{xc}_{pp}	$\delta(CR^{xc}_{pp})$	CR^{xc}_{lz}	$\delta(CR^{xc}_{lz})$
psd7003	10.52	1.0044	9.68	1.0010	8.58	0.9969	**8.53**	1.0156
dblp	13.58	0.9992	10.42	0.9855	**8.93**	1.0050	10.74	1.0105
wiki	17.58	0.9301	14.71	0.9417	**12.61**	0.9498	12.65	0.9467
SwissProt	6.83	0.9238	5.54	0.9560	**4.67**	0.9687	5.77	0.9529
nasa	12.22	0.9793	10.32	0.9947	**8.60**	1.0001	9.58	0.9970

TABLE III
NOTATION USED IN COMPRESSION EXPERIMENTS

Symbol	Meaning	Units
S_0	size of original file	bytes
T_c	time of clutering	hh:mm:ss
CS^t_α	size of compressed file using α method and compressed as *text*	bytes
CS^x_α	size of compressed file using α method and compressed using *XMill*	bytes
CR^t_α	compression ratio using α method and compressed as *text* $CR^t_\alpha = \frac{CS^t_\alpha}{S_0} \times 100\%$	percents
CR^x_α	compression ratio using α method and compressed using *XMill* $CR^x_\alpha = \frac{CS^x_\alpha}{S_0} \times 100\%$	percents
CS^{tc}_α	size of compressed file using α method and compressed as *text* with *clustering*	bytes
CS^{xc}_α	size of compressed file using α method and compressed using *XMill* with *clustering*	bytes
CR^{tc}_α	compression ratio using α method and compressed as *text* with *clustering* $CR^{tc}_\alpha = \frac{CS^{tc}_\alpha}{S_0} \times 100\%$	percents
CR^{xc}_α	compression ratio using α method and compressed using *XMill* with *clustering* $CR^{xc}_\alpha = \frac{CS^{xc}_\alpha}{S_0} \times 100\%$	percents
$\delta(CR^{tc}_\alpha)$	improvement of compression ratio for given CR $\delta(CR^{tc}_\alpha) = \frac{CR^{tc}_\alpha}{CR^t_\alpha}$	1
$\delta(CR^{xc}_\alpha)$	improvement of compression ratio for given CR $\delta(CR^{xc}_\alpha) = \frac{CR^{xc}_\alpha}{CR^x_\alpha}$	1
	where $\alpha \in \{gzip, bzip2, ppm, lzma\}$	

TABLE V
XML COMPRESSED WITHOUT CLUSTERING USING XMILL

(a) Size of Compressed XML

File	CS^x_{gz}	CS^x_{bz}	CS^x_{pp}	CS^x_{lz}
psd7003	75,091,691	69,335,514	61,688,477	**59,119,672**
dblp	97,079,361	75,526,149	**63,497,428**	73,689,071
wiki	102,427,586	84,621,483	**71,919,990**	83,418,527
swissProt	8,492,438	6,652,068	**5,529,656**	6,305,849
nasa	3,126,312	2,597,705	**2,153,068**	2,434,853

(b) Compression Ratio

File	CR^x_{gz}	CR^x_{bz}	CR^x_{pp}	CR^x_{lz}
psd7003	10.48	9.67	8.61	**8.25**
dblp.xml	13.59	10.57	**8.89**	10.32
wiki.xml	18.90	15.62	**13.27**	15.39
SwissProt.xml	7.40	5.79	**4.82**	5.49
nasa.xml	12.48	10.37	**8.59**	9.72

amount of text stored in IRS and auxiliary data structures constitute a suitable material for data compression. However, the data that form the textual database of every IRS are very mixed and it is therefore useful to study special data compression methods.

This paper focuses on improvement of compression of XML documents based on clustering and rearranging of XML elements within XML documents. Such transformed XML documents can be more efficiently compressed. The principle of improving the compression of XML is rearranging elements in order to elements with similar content is move as close as possible to each other. In this way compression algorithm is allowed to exploit similarity among compressed elements. After transformation, the compression itself is done using conventional, universal compression algorithms or using specialized XML compression tools such as XMill. Experimental results show that this method has a positive impact across the tested compression methods. This kind of XML document transformation is suitable only for documents were rearranging of elements is possible and is not prohibited by semantics of the document. The proposed methods has another advantage. It is not necessary to change used compression algorithms, the transformation is made just before compression starts.

ACKNOWLEDGEMENT

This work is partially supported by Grant of Grant Agency of Czech Republic No. 205/09/1079.

REFERENCES

[1] Xgrind: A query-friendly xml compressor. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 225–, Washington, DC, USA, 2002. IEEE Computer Society.

[2] J. Adiego, G. Navarro, and P. de la Fuente. Using structural contexts to compress semistructured text collections. *Inf. Process. Manage.*, 43:769–790, May 2007.

[3] B. T. Al-Hamadani, R. F. Alwan, and J. Lu. Xqpoint: a queriable homomorphic xml compressor. In *Proceedings of the 6th international conference on Innovations in information technology*, IIT'09, pages 26–30, Piscataway, NJ, USA, 2009. IEEE Press.

[4] D. Blandford and G. Blelloc. Index compression through document reordering. In *Data Compression Conf.*, pages 342–351, UT, USA, April 2002.

[5] J. Cheng and W. Ng. Xqzip: Querying compressed xml using structural indexing. In *In International Conference on Extending Database Technology*, pages 219–236, 2004.

[6] J. G. Cleary, Ian, and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32:396–402, 1984.

[7] L. Denoyer and P. Gallinari. The Wikipedia XML Corpus. *SIGIR Forum*, 2006.

[8] J. Dvorský and J. Martinovič. Improvement of text compression parameters using cluster analysis. In V. Snášel, J. Pokorný, and K. Richta, editors, *DATESO*, volume 235 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[9] J. Dvorský, J. Martinovič, and V. Snášel. Query expansion and evolution of topic in information retrieval systems. In V. Snášel, J. Pokorný, and K. Richta, editors, *DATESO*, volume 98 of *CEUR Workshop Proceedings*, pages 117–127. CEUR-WS.org, 2004.

[10] C. Faloutsos. Fast searching by content in multimedia databases. *IEEE Data Eng. Bull.*, 18(4):31–40, 1995.

[11] R. L. Haskin. Special-purpose processors for text retrieval. *Database Engineering*, 4(1):16–29, 1981.

[12] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[13] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[14] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[15] W. Li. Xcomp: An xml compression tool. Technical report, 2003.

[16] H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. *SIGMOD Rec.*, 29:153–164, May 2000.

[17] J. Martinovič and P. Gajdoš. Vector model improvement by fca and topic evolution. In K. Richta, V. Snášel, and J. Pokorný, editors, *DATESO*, volume 129 of *CEUR Workshop Proceedings*, pages 46–57. CEUR-WS.org, 2005.

[18] J. Martinovič, T. Novosad, and V. Snášel. Vector model improvement using suffix trees. In *ICDIM*, pages 180–187. IEEE, 2007.

[19] J.-K. Min, M.-J. Park, and C.-W. Chung. Xpress: a queriable compression for xml data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 122–133, New York, NY, USA, 2003. ACM.

[20] C. Nevill-Manning, Ian, and I. Witten. Compression and explanation using hierarchical grammars. *Computer Journal*, 40:103–116, 1997.

[21] S. Orlando, R. Perego, and F. Silvestri. Assigning document identifiers to enhance compressibility of fulltext indices. In *In SAC'04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 222–229. ACM Press, 2004.

[22] J. Platoš, J. Dvorský, and J. Martinovič. Using clustering to improve WLZ77 compression. In *ICADIWT 2008. First International Conference on Applications of Digital Information and Web Technologies,*, pages 308 – 313. IEEE Computer Society, 2008.

[23] S. Sakr. Xml compression techniques: A survey and comparison. *Journal of Computer and System Sciences*, 75(5):303 – 322, 2009.

[24] V. Toman. Syntactical compression of xml data. In *Presented at 16th Intl. Conf. on Advanced Information Systems Engineering (CAiSE04*, 2004.