World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:8, 2008

Compression of Semistructured Documents

Leo Galambos
Department of Software Engineering
Charles University in Prague
Czech Republic
e-mail: leo.galambos@mff.cuni.cz

Jan Lansky
Department of Software Engineering
Charles University in Prague
Czech Republic
e-mail: zizelevak@gmail.com

Katsiaryna Chernik
Department of Software Engineering
Charles University in Prague
Czech Republic
e-mail: kchernik@gmail.com

Abstract**—** EGOTHOR is a search engine that indexes the Web and allows us to search the Web documents. Its hit list contains URL and title of the hits, and also some snippet which tries to shortly show a match. The snippet can be almost always assembled by an algorithm that has a full knowledge of the original document (mostly HTML page). It implies that the search engine is required to store the full text of the documents as a part of the index.

Such a requirement leads us to pick up an appropriate compression algorithm which would reduce the space demand. One of the solutions could be to use common compression methods, for instance gzip or bzip2, but it might be preferable if we develop a new method which would take advantage of the document structure, or rather, the textual character of the documents.

There already exist a special compression text algorithms and methods for a compression of XML documents. The aim of this paper is an integration of the two approaches to achieve an optimal level of the compression ratio.

Keywords**—** Compression, search engine, HTML, XML.

I. Motivation

EGOTHOR [7] is a full-text search engine written entirely in Java2. The platform was chosen for its beneficial attributes: portability, simplified code management and fast linking with modules of 3rd parties. The issue discussed in this paper is related to the development of a proper compression algorithm with respect to the amount of data processed by the whole system. This problem will not be discussed in a context of the inverted index which is already compressed. Our goal is pointed to the database of original documents in particular. This meta-data database plays an important role during the process of snippets generation. Moreover, it is also a major consumer of a disk space in the whole system.

To better explain our motivation a brief system performance is given. The second generation system (EGOTHOR v2) consists of a Web robot and indexing and search modules. The robot is able to crawl the Web at the speed of 700-1000 pages per second. The indexer throughput is about 500-700 pages per second and the searcher is only limited by the disk bus capacity. Obviously, all the performance values are influenced by a length of documents, structure of the Web and hardware capacity (the values were measured on a dedicated server with AMD Opteron 246).

The work was supported by the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation".

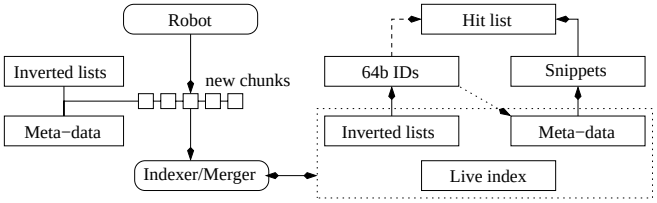

Fig. 1 EGOTHOR architecture

The system can also be extended by many linguistic modules which increase the information quality of results and lowers the throughput consequently. Such modules could also lower the system space demand, i.e. a stemmer [6], but we rather consider now that the system runs without any supplemental modules.

Snippets can be generated by many algorithms, but the common factor is the knowledge of the original document structure and user's query. First, the system finds hits, or rather their 64-bit identifiers, and retrieves their textual contents from the database of original documents. Finally, it prepares adequate snippets for the final hit list. This common process is also described by Figure 1.

The base version of the system implemented the compression algorithm of the database on top of the existing gzip method [5]. The question is whether such a compression is suitable for HTML documents or an adequate substitution could be found on a base of other (or similar) compression methods.

Our aim is therefore to develop such a method that gives a better or similar compression ratio and a same or better decompression time than gzip. On the other hand, the compression time need not be as fast as gzip.

This paper is organized as follows. A brief introduction to the compression is presented first. Then we discuss the textual methods and the methods suitable for XML compression. A novel method for HTML (or XML) documents is introduced afterwards. Finally, experiments and a list of open problems are presented, and the conclusion is given.

II. Compression

Any Web full-text system gathers a lot of documents, mostly HTML pages [26]. The number of all pages is estimated over several billions and their average size is about 10-20kB [18].

The capacity issue can be then solved by a compression which is a typical way how to reduce the data size.

A successful compression method can take advantage of the knowledge of the input message (stream). Our document database largely contains HTML documents and it is a good idea to make the best of HTML tree structure.

Obviously, we could still forget all the HTML features and pick up one of the many compression methods, such as gzip or bzip2 [20], running on a textual representation of the input documents. The compression level might be good, but also suboptimal in a common case. This solution was chosen for its simplicity by the existing EGOTHOR system.

Another way is to use special compression algorithms which process the textual representation by syllables [13] or words [24]. The syllable-based methods often need to know a language of the input document, but it is not a serious issue. The language can be found in HTTP header or HTML meta-data values, or even guessed by some statistical analysis of the text.

There is also another way how to compress the HTML documents. Existing methods for XML compression [25] could be used, for instance XMLPPM [8] which is thought to be the most effective method for XML data. This sort of algorithms is based on an effective coding of XML tags structure. Unfortunately, HTML standard is not as strict as XML and it makes the use of tag structure harder. We would have to fix our input HTML documents to well-formed XML, or extend the original XML compression methods to support documents which are not well-formed.

The original XML-based solution tries to separate two processes. The first one encodes the tags and attributes structure, while the second one encodes the CDATA content with some common compression method. Some improvement was achieved by the use of a different compression method for the second process [9]. This paper presents a similar approach with the extension for HTML documents.

III. Compression Methods for Text

Text compression can often derive benefit from the two views of the textual content: the content can be seen as a stream of syllables or words. The word-based methods are older, so many implementations of classical methods exist, for instance Huffman coding [24], LZW [4], Burrows-Wheeler transformation [10], PPM [1] or Arithmetic coding [17]. The syllable-based methods are rather young with initial implementations of Huffman coding and LZW [13].

The port of classical character methods to syllable or word based is not easy. The transformation heavily hits almost all inner data structures, because they must be able to work with undefined (and often high) number of syllables or words instead of the original alphabet of 256 characters. Moreover, the large input alphabet (of syllables or words) also requests the encoder to export elements of the alphabet to the decoder. This issue is often solved by exporting the alphabet as a part of the encoded document [11].

The confrontation and comparison of the word and syllable based methods depends on a language of the input document. Languages with a simple morphology, i.e. English, are better compressed by the word-based algorithms. On the other hand, the languages with a complex morphology, i.e. Czech or German, are often compressed better by the syllable-based methods [12].

A. Word-Based Methods

The word-based methods require to divide the input document into a stream of words and non-words. The words are usually defined as longest alphanumeric strings in a text, while the non-words are the remaining fragments.

The previous definition of words and non-words implies that one can assume that the elements of the two groups are alternated regularly. Next, another heuristic is also used: the word is often followed by a special non-word – space. So we can skip over the space without any encoding. The right decompression is guaranteed, one must only ensure that two successive words are interleaved with the space in a decoder.

In practice the words length is often limited by some constant value. Longer words are broken up and the resulting parts are interleaved with a special empty word of an opposite type (word versus non-word). For instance, if a long word is divided into two parts, then the parts are interleaved with an empty non-word.

B. Syllable-Based Methods

The syllable-based methods [13] first decompose an input document into words and then these words are decomposed into the final stream of syllables. The previous word based methods recognize two word groups. In contradistinction to this simple approach, the syllable-based methods prefer the following (more effective) grouping.

Words containing only the small letters are denoted as *small*, i.e. “river”. Words containing only the upper letters are *upper*, i.e. “MSFT”. Words starting with first letter upper and having following letters small are *mixed*, i.e. “John”. If a word contains digits and nothing else, then the word is classified as *number*, i.e. “2006”. Other non-alphanumeric words form the last catch-all group – *special*. Moreover, the first three groups are also named *letter* words, the last two groups are denoted as *non-letter*.

The input textual document is decomposed into words by a greedy algorithm. Afterwards, the words are decomposed into syllables. This process need not be always unambiguous. In our case, we can be satisfied with some approximation, because it has not any significant impact on the final compression ratio.

There are many algorithms which are able to decompose words into syllables. We briefly present four algorithms which need to know a few linguistic rules about the processed language. In fact, they only need to recognize vowels and consonants correctly.

All the four algorithms has a common start phase. All non-letter words are declared as syllables, and therefore they are

TABLE I
HYPHENATION OF *odstrčenou* (CZECH WORD: "UNDERPRIVILEGED")

Algorithm	Syllables
Correct hyphenation	od-str-če-nou
P_{UL}	odst-rč-en-ou
P_{UR}	o-dstr-če-nou
P_{UML}	ods-tr-če-nou
P_{UMR}	od-str-če-nou

not divided at all. In letter words, vowels and consonants are recognized. Next, the longest strings of vowels are found. The string (block) must be of the maximum length of 3 letters and it cannot be possible to extend it by a contiguous vowel. These blocks are cores of final syllables. Consonants before the first block are assigned to the first block, and consonants after the last block are assigned to the last block.

The algorithms differ in the following step which assigns remaining consonants to the existing blocks which are left or right to them. The universal left algorithm (P_{UL}) assigns all consonants to the left (previous) block of vowels. Similarly, the universal right algorithm (P_{UR}) assigns all consonants to the right block.

The universal middle-right algorithm (P_{UMR}) assigns one half of consonants to the respective adjacent block, if the number of consonants is even. If the number is odd, then the right block gets one consonant more.

Last, the universal middle-left algorithm (P_{UML}) is antipodal – it prefers to assign one consonant more to the left block. Nonetheless, one exception exists: if there is just one consonants, then it is assigned to the right block. It ensures that word endings are not formed by single blocks of vowels, which is the handicap of the P_{UL} algorithm.

An example is presented in Table I, where all the four algorithms hyphen a word *odstrčenou* (Czech word: "underprivileged"). The blocks of vowels are *o*, *r*, *e*, *ou* (in this order).

The example also presents a surprising fact – the recognition of vowels and consonants is not so simple in many languages. Basic vowels are obvious, there are *a*, *e*, *i*, *o*, *u*, *y*, while other letters are called basic consonants. Nonetheless, a context and language can cause that some basic vowels may start to play a role of consonants (and vice versa). For instance, consonants *r* and *l* play a role of vowels in Czech, if their adjacent letters are consonants. A similar case can be shown with the letter *y* in English. It depends on a context which role the letter *y* plays, it can be a vowel, i.e. *dirty*, or a consonant, i.e. *yellow*.

The syllable-based methods are based on a fact, that the text consists of sentences and it can be described by the following rules: A sentence starts with a mixed word (first letter is upper, others are small) and ends with a special word containing a dot. The small and special words are alternated regularly. If a sentence starts with an upper word, then the upper and special words are alternated instead.

This model does not work well after the hyphenation process. Every word has a different number of syllables and it may cause the following issue. While a small word is often followed by a special word, a small syllable can be followed by a small syllable (and obviously by a special syllable as well).

This observation helps to define a model which is able to predict the type of a next syllable. More details can be found in the previous paper [13] on this topic.

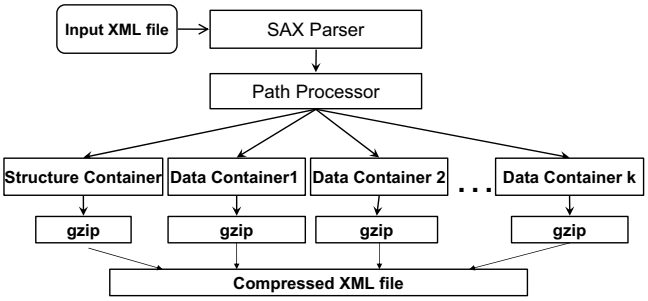

Fig. 2 Data flow of XMill

IV. XML STRUCTURE COMPRESSION

There are many algorithms which compress XML data. One of the first available was XMill [14]. Many other successors are based on similar principles, i.e. XMLPPM [8]. Some algorithms also add new features: XGrind [21] and XPress [16] are able to query the compressed data structure at the cost of a worse compression ratio.

A. XMill

XMill algorithm is based on the following principles:

- Structure separation: The structure includes tags, attributes and their ordering. Sequence of fields (textual content of tags, values of attributes) is rated as data.
- Grouping of data: Data values can be grouped into containers by their sense and each of the groups is compressed separately. For instance, one container can be formed by a content of all `<name>` tags, while the second container is formed by a content of `<phone>` tags.
- Separate compression of containers: Every container is often processed by a different compression method.

The data flow of XMill method is described by Figure 2. The input document is processed by SAX parser [15] which forwards SAX events to Path processor. The processor separates structure and data, and also groups data into containers.

There is one special container which processes the structure of the whole document. Its data stream has the following format: tag content is replaced by a number of a container which processes the given tag; tag names and attributes are replaced by references to a vocabulary of tags and attributes.

Finally, the content of all containers is compressed by gzip, and the resulting stream is saved to a final output.

B. XMLPPM

XMLPPM [8] is XML compressor based on Prediction by Partial Match (PPM) encoding. It proposes a technique called

Multiplexed Hierarchical Modeling (MHM), which employs two basic ideas: *multiplexing* several text compression models based on XML syntactic structure, and *injecting* hierarchical element structure symbols into the multiplexed models.

In XMLPPM, the input XML document is converted into a stream of SAX events. Element start tags, end tags, and attribute names are dictionary encoded and sent to corresponding PPM models for running predictions and encodings.

XMLPPM uses four compression models:

1) the element and attribute name model (Sym),
2) the element structure model (Elt),
3) the attribute values model (Att), and
4) the string value model (Char).

To illustrate the operation of XMLPPM, consider this XML fragment: `<elt att="abcd">XYZ</elt>`. Assuming the tag `elt` has been seen before, and is represented by byte `10`, while the attribute name `att` has not, and the next available byte for attribute name is `0D`, our XML fragment would be encoded as shown in Figure 3.

In Att and Char models, XMLPPM injects the enclosing token index `<nn>` in order to retain cross-model dependencies among the tokens in different contexts. The `<nn>` token indicates that a particular token has been seen, but these token indexes are not explicitly encoded in the models.

Last, but not least, XMLPPM is often able to achieve a better compression ratio than the default mode of XMill [14].

V. Our Contribution

This section presents our older methods XMillSyl and XMLSyl [9] which were used to test the joint of syllable-based methods with some XML structure compression methods.

A. XMillSyl and XMLSyl

We already made a proposal of two methods which try to combine the compression of XML structure and text.

The first method (XMillSyl) is a modification of XMill algorithm. While the structure container is still compressed by gzip, our method uses syllable-based compression methods (LZWL and HuffSyllable [13]) instead of the original gzip.

The second method (XMLSyl) tries to modify the existing syllable-based methods LZWL and HuffSyllable, so that the tags are not hyphened. There is still the SAX parser which sends SAX events into a structure coder. The coder uses two separate dictionaries for tags and attributes encoding, and it also replaces tag and attribute names with references to the respective dictionary. This output stream is processed by a data container. The data container and dictionaries are finally encoded by LZWL or HuffSyllable.

Both methods (LZWL and HuffSyllable) are syllable-based methods, but their roots differ. LZWL is based upon a classic dictionary-based LZW [23] algorithm, while HuffSyllable is inspired by HuffWord [24] and uses the principle of the five different syllables.

B. XBW

The new XBW method is based on Burrows-Wheeler transformation [3]. Burrows-Wheeler transformation was chosen for its success in the bzip2 program [20].

The method consists of these seven steps: replacement of tag names, division into words or syllables, dictionary encoding, Burrows-Wheeler transformation (BWT), Move to Front transformation (MTF), Run Length Encoding of null sequences (RLE), Canonical Huffman.

1) Replacement of tag names: The XBW method SAX parser produces a sequence of SAX events which are processed by a structure coder. The coder builds up two separate dictionaries for tags and elements encoding. Moreover, it also replaces tag and attribute names with references to the respective dictionaries.

2) Division into words or syllables: The output of the previous step is divided into words or syllables as described in Section III. The resulting stream is denoted as S-stream.

3) Dictionary encoding: The previous step also generates a dictionary of words or syllables which are used in a text. One of the effective dictionary compression methods is TD3 [11]. The method encodes the whole dictionary (represented as a trie) instead of the separate items stored inside.

4) Burrows-Wheeler transformation: The purpose of BWT step is to transform the S-stream into a "better" stream. The "better" stream would allow to achieve a better compression ratio. Obviously, the transformation would be also reversible else we might lost some information. In the concrete, a partial grouping of same input alphabet elements will be achieved. Such a process requires to sort all the permutations of this step input. We do not yet use the effective algorithm described in [19], but a simpler qsort function of C/C++ language. The sophisticated algorithm would boost the compression time performance.

5) Move to Front transformation: Next, the output stream of BWT is transformed by another transformation step – MTF [2]. This step translates textual strings into a sequence of numbers. Suppose a numbered list of alphabet elements. MFT reads input elements and writes their list order. As soon as an element is processed, it is also moved up to the front of the list.

6) Run Length Encoding of null sequences: MFT step may generate a long sequences of zeroes (null sequences). The successor step (RLE) shrinks the null sequences and replaces them with a special symbol which represents a null sequence of a given length. The output is then a stream of numbers and the special symbols.

7) Canonical Huffman: Finally, the stream after RLE step is encoded by canonical Huffman code [22].

VI. Experiments

This paper discussed several methods which could be used for HTML pages compression. This section presents the respective experiments with the input of ten EGOTHOR robot files. Each of the files has size about 14 MB and contains one thousand HTML pages of *ac.uk* domain. The HTML pages

Model	<elt	att=	"asdf"	>	XYZ	</elt>
Elt:	10				FE	FF
Att:		<10> 0D	asdf 00	<10> FF		
Char:					<10> XYZ 00	
Sym:		att 00				

Fig. 3 Example: XMLPPM processing

TABLE II
RESULTS OF COMPRESSION (10 FILES, 1000 ENGLISH PAGES EACH)

Method	Compressed	Ratio
none	14 383 kB	100.00%
gzip	3 044 kB	21.16%
bzip2	2 372 kB	16.50%
XMLPPM	2 128 kB	14.80%
XBW (words)	1 379 kB	9.59%
XBW (syllables)	1 371 kB	9.53%

were first fixed and repaired to comply the well-formed XML format. The summary results (average per one file) are then presented in Table II.

The first column of the table specifies the method used, the next column presents the size of the original file after the compression, and the last column shows the compression ratio achieved.

The most important factor is the compression ratio, and it can be seen that the absolute winner is XBW with the ratio of 9.53%. It implies, that the original file occupies less than one tenth of its original size. XMLPPM output needs about 50% more space (compared to XBW), bzip's about 80% more and gzip's about 120% more.

An interesting point can be seen on the results of word and syllable-based variation of XBW as well. Although the syllable-based methods are worse than word-based, in this case of a language of a simple morphology (English), they are slightly better with XBW core. We suppose even better result on languages with a rich-morphology, thus we may simply prefer the syllable-based method later.

Last, but not least, another factor is important as well, it is a time of compression. The fastest program is gzip with 2 seconds per our file. Bzip runs slower and needs 9 seconds. XMLPPM ends its work after 20 seconds.

The XBW program exists in a beta version and it is not fully optimized for speed yet. Obviously, it is the slowest among the tested algorithms - one run needs about 400 seconds, but we suppose 50-100 seconds after a full optimization. On the other hand, the unoptimized XBW achieves a decompression time better than 60 seconds. Such a result is good enough for our primary use in a beta version of a search engine.

The results show that the existing use of gzip is not suitable. GZip is fastest, but it also achieved the worst compression ratio in out test group of five. Unfortunately, the method with the best compression ratio (XBW) is not still ready to go live down to earth – it needs significant optimization first.

VII. OPEN PROBLEMS

The methods based on XML structure compression require well-formed input documents. Unfortunately, this requirement is often contravened in HTML documents on the Web: the documents use tags incorrectly, i.e. <a>, or their tags use the same attribute name twice. Therefore, it will be needed to modify the existing XML structure compression method to support bad-formed documents at the cost of lowering a compression ratio.

Text compression methods, especially syllable-based, are able to take advantage of the language specification given by HTTP headers or HTML meta-data block. In practice we found that the specification was often false or it could not cover a situation when the document contained paragraphs or sentences in different languages.

On the other hand, HTML standard has one positive aspect – it uses a limited set of tags and attributes names. This aspect may improve the efficiency of a compression process.

A negative aspect of HTML could be seen in a support of scripting languages. Since the scripts have not a structure of a natural language, the textual compression methods do not achieve the best results here.

Presently, EGOTHOR robot stores up to one thousand HTML pages into one single file. The reasons are rooted in the robot optimizations. Unfortunately, it also implies that one cannot easily access single documents without the decompression of all documents in this file. We already plan to compress the single documents, so that the final file would contain 1000 documents compressed separately. It would ease the access to the single documents. Such a solution may utilize the syllable-based methods effectively [12].

VIII. CONCLUSION

EGOTHOR full-text search engine collects and stores huge number of documents, mostly in HTML format. For practical reasons the documents are compressed obviously.

We discussed the selection of a suitable compression method which would utilize the semantics and structure of HTML documents. Our guess was that such a method has the best chance to achieve an optimal level of a compression ratio.

Three branches of compression algorithms were discussed: textual, special XML, and a mix of the previous two. Last branch was represented by a novel XBW algorithm which combines textual method with a method for XML structure compression.

REFERENCES

[1] Adiego, J., Feunte, P.: On the Use of Words as Source Alphabet Symbols in PPM. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA (2006) 435.
[2] Arnavut, Z.: Move-to-front and inversion coding. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA (2000) 193–202.
[3] Burrows, M., Wheeler, D. J.: A Block Sorting Loseless Data Compression Algorithm. Technical report, Digital Equipment Corporation, Palo Alto, CA, U.S.A (2003).
[4] Dvorsky, J., Pokorny, J., Snasel, V.: Word-based Compression Methods for Large Text Documents. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA (1999) 523.
[5] Gailly, J. L.: Gzip program and documentation (1993). Source code aviable from ftp://prep.ai.mit.edu/pub/gnu/gzip-*.tar
[6] Galambos, L.: Dynamization in IR Systems. Mieczyslaw A. Klopotek, Slawomir T. Wierzchon, Krzysztof Trojanowski (Eds.): IIPWM, Proc. of the Int. IIS: IIPWM'04, Poland, 2004. ASC Springer 2004, ISBN 3-540-21331-7.
[7] Galambos, L.: EGOTHOR. http://www.egothor.org/
[8] Cheney, J.: Compressing XML with Multiplexed Hierarchical PPM Models. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA (2001) 163.
[9] Chernik, K., Lansky, J., Galambos, L.: Syllable-based compression for XML documents. In: Snasel, V., Richta, K., and Pokorny, J.: Proceedings of the Dateso 2006 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. **176**, (2006) 21-31
[10] Isal, R.Y.K., Moffat, A.: Word-based Block-sorting Text Compression. Proc. 24th Australasian Computer Science Conference, Gold Coast, Australia, (2001) 92–99
[11] Lansky, J., Zemlicka, M.: Compression of a Dictionary. In: Snasel, V., Richta, K., and Pokorny, J.: Proceedings of the Dateso 2006 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. **176**, (2006) 11-20
[12] Lansky, J., Zemlicka, M.: Compression of Small Text Files Using Syllables. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA (2006) 458.
[13] Lansky, J., Zemlicka, M.: Text Compression: Syllables. In: Richta, K., Snasel, V., Pokorny, J.: Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. **129**, (2005) 32–45
[14] Liefke, H., Suciu, D.: XMill: an Efficient Compressor for XML Data. In Proc. ACM SIGMOD Conference (2000) 153–164
[15] Megginson, D.: SAX: A Simple API for XML. http://www.saxproject.org
[16] Min, J. K., Park, M. J., Chung, C. W.: XPRESS: A Queriable Compression for XML Data. SIGMOD 2003, San Diego, CA, USA (2003) 122–133
[17] Moffat, A., Neal, R. M., Witten, I. H.: Arithmetic Coding Revisited. ACM Transactions on Information Systems, **16**, (1998) 256–294
[18] O'Neill, E. T., Lavoie, B. F., Bennett, R.: Trends in the Evolution of the Public Web: 1998–2002. D-Lib Magazine (2003) 1082–9873
[19] Seward, J.: On the Performance of BWT Sorting Algorithms. DCC, IEEE CS Press, Los Alamitos, CA, USA (2000) 173.
[20] Seward, J.: The bzip2 and libbzip2 official home page. http://sources.redhat.com/bzip2/
[21] Tolani, P., Haritsa, J. R.: XGrind: A Query-friendly XML Compressor. In Proc. IEEE International Conference on Data Engineering (2002).
[22] Turpin, A., Moffat, A.: Housekeeping for prefix coding. IEEE Transaction on Communications, 48(4), (2000) 622–628.
[23] Welch, T. A.: A technique for high performance data compression. IEEE Computer, **17(6)** (1984) 8–19.
[24] Witten, I., Moffat, A., Bell, T.: Managing Gigabytes: Compressing and Indexing Documents and Images. Van Nostrand Reinhold (1994).
[25] World Wide Web Consorcium: Extensive Markup Language (XML). http://www.w3.org/XML/
[26] World Wide Web Consorcium: HyperText Markup Language (HTML). http://www.w3.org/MarkUp/