Syllable-based Compression for XML Documents

Katsiaryna Chernik, Jan Lánský, and Leo Galamboš

Charles University, Faculty of Mathematics and Physics
Department of Software Engineering
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
kchernik@centrum.cz, zizelevak@gmail.com, leo.galambos@mff.cuni.cz

Abstract. Syllable-based compression achieves sufficiently good results on text documents of a medium size. Since the majority of XML documents are of that size, we suppose that the syllable-based method can give good results on XML documents, especially on documents that have a simple structure (small amount of elements and attributes) and relatively long character data content.
In this paper we propose two syllable-based compression methods for XML documents. The first method, XMLSyl, replaces XML tokens (element tags and attributes) by special codes in input document and then compresses this document using a syllable-based method. The second method, XMillSyl, incorporates syllable-based compression into the existing method for XML compression XMill. XMLSyl and XMillSyl are compared with a non-XML syllable-based method and with other existing method for XML compression.

1 Introduction

The Extensible Markup Language (XML) [5] is a simple text format for structured text documents. XML provides flexibility in storing, processing and exchanging data on the Web. However, due to their verbosity, XML documents are usually larger in size than other exchange formats containing the same data content. One solution to this problem consists of compressing XML documents. Because XML is a text format, it is possible to compress XML documents with existing text compression methods. These methods are more effective, when XML documents have simple structure and long character data content. There are different types of text compression: text compression by characters and text compression by words. There is also a novel method: text compression by symbols [12]. In our work an application of this method to XML documents was developed. Since single text compression is not able to discover and utilize the redundancy in the structure of XML, we modify syllable-based compression method for XML.

At the beginning we supposed that XML syllable-based compression will be suitable for middle-sized textual XML documents. There are many XML documents that meet these conditions, for example any documentation written in DocBook [16] format or news in RSS format [18]. Moreover we suppose that our compression would be more suitable for documents in languages with rich morphology (for example Czech or German [12]).

2 Syllable-based compression method

Syllable-based compression [12] is the method where compression is performed at the syllable level. There are two syllable-based compressors. The first one is syllable-based LZW, and the second one is syllable-based Huffman.

Algorithm LZW [11] is a dictionary compression character-based method. The syllable-based version is called LZWL. In the initialization step, the *syllable dictionary* is filled with empty syllable and syllables from a database of frequent syllables. The following steps are similar with character-based version of LZW, but LZWL works over an alphabet of syllables.

The second syllable-based compression method is called HuffSyllable. It is a statistical compression method based on the adaptive Huffman coding. For our purposes, we use only LZWL syllable-based compression method. Adaptation of HuffSyllable for XML compression gave worse results than LZWL.

3 XMLSyl

Our goal was to modify the syllable-compression method to compress XML documents efficiently. We attempted to modify existing syllable-based compressor so, that it treats XML tokens (element tags and attributes) as single syllables instead of decomposing them into many syllables. There were two possibilities to compel the syllable-based compressor to treat XML tokens as syllables:

1. Modify parser used in the syllable-based tool and combine it with an XML parser, so that it can recognize XML tokens and treat them as a single syllable.
2. Replace XML tokens with bytes in the input document and then compress such a document with an existing syllable-based tool.

We decided to implement the second way because this implementation allows us to make some future improvements easily. For example, we may compel the syllable-based compressor to assign codes with minimal length to XML tokens by adding this single bytes to the *syllable dictionary*[12]. This improvement is impossible in the first variant. The encoding of XML tokens is inspired by existing XML compression methods like XMLPPM [3], XGrind [6], XPress [9], XMill [8].

3.1 Architecture and principles of XMLSyl

The architecture of XMLSyl is shown in Figure 1. It has four major modules: the *SAX Parser*, the *Structure Encoder*, the *Containers* and the *Syllable Compressor*. First, the XML document is sent to the SAX Parser. Next the parser decomposes document into SAX events (start-tags, end-tags, data items, comments and etc.) and forwards them to the Structure Encoder.

The Structure Encoder encodes the SAX events and routes them to the different Containers. There are three containers in our implementation:

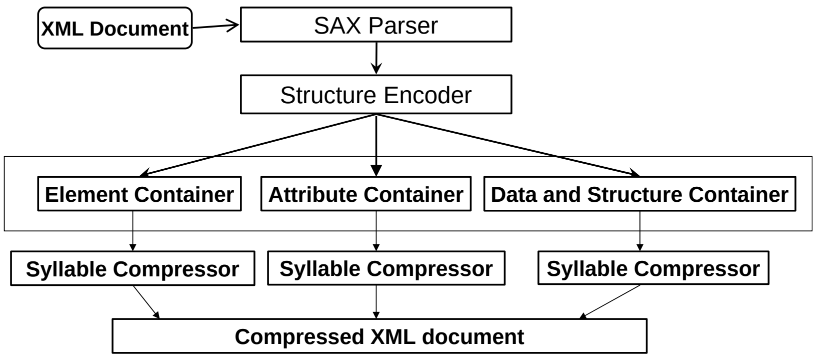

Fig. 1. The Architecture of XMLSylCompressor

1. **Element Container:** The Element Container stores the names of all elements that occur in an XML document. The Structure Encoder also uses the Element Container as the dictionary for encoding XML structure.
2. **Attribute Container:** The Attribute Container stores the names of all attributes which occur in an XML document. The Structure Encoder also uses the Attribute Container as the dictionary for encoding XML structure.
3. **Structure and Data Container:** The Structure and Data Container stores an XML document, in which all meta-data are replaced with special codes. The encoding process is presented in section 3.2.

When a document is parsed and separated into the containers completely, the contents of the containers are sent to the Syllable Compressor. It compresses the content of each container separately using syllable-based compression and sends the result to the output.

We have not written the SAX parser by ourselves, rather we have used the Expat parser[10] which is an open-source SAX parser written in C.

3.2 Encoding the structure of XML document

The structure of XML document is encoded in XMLSyl as follows. Whenever a new element or attribute is encountered, its name is sent to the dictionary and the index of the element is sent to the Data and Structure Container. Two different dictionaries are used for attributes and elements: the Element Dictionary and the Attribute Dictionary. The Attribute Container operates as the Attribute Dictionary and the Element Container as the Element Dictionary. Whenever an end tag is encountered a token `END_TAG` is sent to the Data and Structure container. Whenever a character sequence is encountered, it is sent to the Data and Structure Container without changes. Start and end of character sequences are indicated by special tokens. We distinguish four different character sequences:

value of attribute, value of element, comment, and white spaces between tags, if white spaces are preserved.

To illustrate the encoding process, consider the encoding of the following small XML document:

```
<book>
  <title lang="en">XML</title>
  <author>Brown</author>
  <author>Smith</author>
  <price currency="EURO">49</price>
</book>
<!-- Comment-->
```

First, the XML document is converted into a corresponding stream of SAX events:

```
startElement("book")
startElement("title",("lang","en"))
characters("XML")
endElement("title")
startElement("author")
characters("Smith")
endElement("author")
startElement("author")
characters("Brown")
endElement("author")
startElement("price","currency","EURO")
characters("49")
endElement("price")
endElement("book")
comment("Comment")
```

The tokens in the SAX event stream are sent to the Structure Encoder. It encodes them and sends them to their corresponding containers. When the *book* start element token is encountered, the string *book* is sent to the Element Container since this element name was not encountered before. An index *E0* is assigned to this entry. This index is sent to the Data and Structure Container. The same operation is executed for *title* start element. String *title* is sent to The Element Container and an index *E1* is assigned to it. The index *E1* is sent to the Data and Structure Container. The element *title* has the attribute *lang*. The attribute name is sent to the Attribute Container and the index *A0* is assigned to it. The index *A0* is sent to the Data and Structure Container. Then attribute value *"en"* is sent without modification to the Data and Structure Container. The *"en"* attribute is followed by the token `END_ATT`, that signals the end of the attribute value. When an element value such as *"XML"* is encountered, the token `CHAR`, signaling the beginning of character sequence, the data value and then the token `END_CHAR` are all sent to the Data and Structure Container. Finally, all the end tags are replaced by the token END_TAG. When a comment event is encountered, the code CMNT is put into the Data and Structure Container. The comment is also sent to the container and is enclosed by END_CMNT code. The final state of all containers is shown in Figure 2.

Element Container

element	index
book	E0
title	E1
author	E2
price	E3

Attribute Container

attribute	index
lang	A0
currency	A1

Data and Structure Container

`<book>`	`<title`	`lang="en">`	XML	`</title>`	`<author>`
E0	E1	A0 en `END_ATT`	`CHAR` XML `END_CHAR`	`END_TAG`	E2

Brown	`</author>`	`<author>`	Smith	`</author>`	`<price`
`CHAR` Brown `END_CHAR`	`END_TAG`	E2	`CHAR` Smith `END_CHAR`	`END_TAG`	E3

`currency="EURO">`	49	`</price>`	`</book>`	`<!--Comment-->`
A1 Euro `END_ATT`	`CHAR` 49 `END_CHAR`	`END_TAG`	`END_TAG`	`CMNT` Comment `END_CMNT`

Fig. 2. Content of containers

In this example we have ignored white spaces between tags, e.g. `<book>` and `<title>`, so the decompressor then produces a standard indentation. Optionally, XMLSyl can preserve the white spaces. In that case, it stores the white spaces as the sequence of characters in the Data and Structure Container between tokens WS and END_WS.

3.3 Containers

The containers are the basic units for grouping XML data. The Attribute Container holds attribute names and the Element Container holds element names. As long as the number of all element and attribute names in any XML document is not high, this two containers are kept in main memory. During parsing, the containers size increases as the container is filled with entries. Each entry in the Element container is assigned a byte in the range 00-A9. These bytes are used for encoding the element names. Each entry in the Attribute container is assigned a byte in the range AA-F9. These bytes are used for encoding the attribute names. The residual 6 bytes are reserved for special codes like CHAR, END_TAG etc. In most cases, 170 (or 80) bytes are enough to encode element (or attribute) names. If the number of elements (or attributes) are greater than 170 (or 80), entries are encoded with two bytes, then tree and so on.

There is another situation with The Data and Structure Container. We do not know the size of the input XML document. The size of XML document can be so big, that document will not fit into memory, and it is not possible to increase the size of container endlessly. Therefore, the container consists of two memory block of constant size. The content of the first memory block is compressed, as soon as the container is filled. We don't compress two blocks at once, because the context of the second memory block is used for compression of the first one. After the compression, the compressed content of the first block is sent to the output and the first block swaps its purpose with the second one. Now the first block is filled with data. When it is full, the second block is compressed, and so on.

3.4 The Syllable Compressor

The Syllable Compressor compresses the Structure and Data Container first and sends the output to the output file. Then the Attribute Containers are compressed and sent to the output file and finally the same happens with the Element Container. LZWL is used for the compression of data. HuffSyll could be also chosen, but the performance is worse, so we decided to use only LZWL.

4 XMillSyl

This chapter introduces our second syllable-based XML compressor, XMillSyl. This second method incorporates syllable-based compression with the existing method for XML compression of XMill [8]. XMill has two main principles in order to optimize XML compression:

– separating structure from data content, and
– grouping Data values with related semantics in the same "container".

Each data container is then compressed individually with gzip [21]. In XMillSyl, containers are compressed with LZWL.

We do not suppose that XMillSyl method gives better results than XMill because gzip compression performs better than LZWL. We have implemented XMillSyl in order to compare the power of XMLSyl with the power of two main principles of XMill.

4.1 Implementation

We did not write XMill compressor. We decided to use existing sources of XMill.

XMill operates as follows: a SAX parser parses the XML file and the SAX events are sent to the core module of the XMill called the path processor. It determines how to map tokens to containers: element tag names and attribute names are encoded and sent to the structure container, while the data values are sent to various data containers, according to their semantic. Finally, the containers are gzipped independently and stored on disk.

We have modified compression and decompression functions (operating on containers) in the way they compress and decompress the data containers with

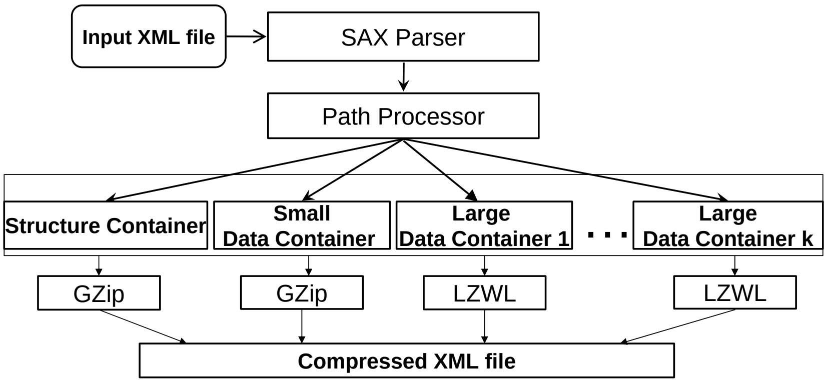

Fig. 3. Architecture of XMillSyl

the syllable-based method (see Figure 3). Moreover we have modified the syllable-based method so that it can work with the containers of XMill implementation instead of a file stream.

XMillSyl discerns the difference between small and large containers. Since LZWL is not suitable for extremely small data, the small containers are compressed with gzip. The structure container is also gzipped in XMillSyl. The large containers are compressed with LZWL.

Table 1. The first data set.

	Size	Lang	Description
elts	103919	English	Periodic table of the elements in XML
pcc	2600257	English	Formal proofs transformed to XML
stats	869059	English	One year statistics if baseball players
tal	1364576	English	Safe-annotated assembly language converted to XML
tpc	313193	English	The XML representation of the TPC_D benchmark database.

5 Comparison Experiments

To show the effectiveness of XMLSyll and XMillSyl, we compared the performance of this two compressors with one representative of XML compressors XMill and the syllable-based compressor LZWL [12].

5.1 XML data sources

XMLSyl and XMillSyl were tested on two data sets that cover a wide range of XML data formats and structures. The first data set is shown in Table 1. It contains English XML documents with different inner structure. It includes regular data that has regular markup and short character data content (elts, stats, weblog, tpc). It also includes irregular data, that has irregular markup (pcc, tall).

The second data set is shown in Table 2. It contains textual XML documents of simple structure with long character data content. It contains five stage plays marked up as XML, four in English and one in Czech. It also contains data in DocBook format in Czech and in English.

Some data was distributed with the XMLPPM [3] and the Exalt [4] compressors while others were found on Internet [15], [16]. All Czech documents use Windows-1250 encoding.

Table 2. The second data set.

	Size	Lan	Description
errors	153530	English	"The Comedy of Errors" marked up as XML
hamlet	314677	English	"The Tragedy of Hamlet, Prince of Denmark" marked up as XML
antony	289865	English	"The Tragedy of Antony and Cleopatra" marked up as XML
much_ado	220495	English	"Much Ado about Nothing" marked up as XML
ch00	13916	English	"DocBook: The Definitive Guide" in DocBook format (1)
ch01	55015	English	"DocBook: The Definitive Guide" in DocBook format (2)
ch02	160728	English	"DocBook: The Definitive Guide" in DocBook format (3)
ch03	27799	English	"DocBook: The Definitive Guide" in DocBook format (4)
ch04	137440	English	"DocBook: The Definitive Guide" in DocBook format (6)
ch05	67142	English	"DocBook: The Definitive Guide" in DocBook format (7)
glossary	24701	English	"DocBook: The Definitive Guide" in DocBook format (8)
howto	42853	English	"DocBook V5.0, Transition Guide" in DocBook format.
hledani	16429	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (1)
komunikace	50881	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (2)
navihace	18495	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (3)
robot	25405	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (4)
xml	28467	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (5)
rur1	59609	Czech	"R.U.R" marked up as XML.

5.2 Compression Performance Metrics

The compression ratio is defined as follows:

$$CR = \frac{sizeof(compressed\,file) \times 8}{sizeof(original\,file)}\ bits/byte.$$

We compare XMillSyl and XMLSyl compression ratios with those of XMill. The compression ratio factor shows normalization of the compression ratio of

Table 3. The first data set.

	CR_{LZWL}	CR_{Xmill}	$CR_{XMillSyl}$	$CRF_{XMillSyll}$	CR_{XMLSyl}	CRF_{XMLSyl}
elts	1,04	0,47	0,54	**1,15**	0,72	**1,53**
pcc	0,22	0,02	0,03	**1,50**	0,04	**2,00**
stats	0,67	0,33	0,40	**1,21**	0,39	**1,18**
tal	0,36	0,09	0,12	**1,33**	0,15	**1,67**
tpc	1,82	1,05	1,54	**1,47**	1,60	**1,52**
Average	0,82	0,39	0,53	**1,33**	0,58	**1,58**

XMillSyll or XMLSyl with respect to XMill. The compression ratio factor is defined as follows:

$$CRF_{XSyl} = \frac{CR_{XSyl}}{CR_{XMill}}.$$

5.3 Experimental Results

The compression ratio statistics of two sets of XML documents are shown in Table 3 and Table 4.

The syllable-based method performed worse on documents from the first data set. On the other hand, both XMLSyl and XMillSyl shows great improvement comparing to LZWL. They compressed the input to 50-60% of the size of the compressed file with LZWL.

On XML documents of the second data set, LZWL provides a reasonably good compression ratio - on the average, about two-thirds that of XMill. This confirms our prediction, that syllable-based compression is effective for textual XML documents. Moreover our compression methods show even greater improvement.

On the document of the second data set, XMillSyl achieves about 15% and XMLSyl is about 20% better compression ratio than LZWL. Compared to XMill, both methods perform slightly worse. XMillSyl compresses about 13% and XMLSyl about 7% worse than XMill.

Figure 4 shows the variation of the compression ratio as a function of XML data size for "DocBook: The Definitive Guide".The compression was run on several subsets. On small files XMillSyl performs better than XMLSyl. The explanation is, that the data are split into many small containers in XMillSyl, which are compressed with gzip (gzip outperforms LZWL, especially on small data). On middle-sized and large files XMLSyl outperforms XMillSyl. We can observe that the bigger size also implies a better compression.

6 Conclusion

In this work we introduced syllable-based compression tools for XML documents called XMLSyl and XMillSyl. We presented the architecture and implementation

Table 4. The first data set.

	CR_{LZWL}	CR_{Xmill}	$CR_{XMillSyl}$	$CRF_{XMillSyll}$	CR_{XMLSyl}	CRF_{XMLSyl}
errors	1,98	1,83	2,00	**1,09**	1,83	**1,00**
hamlet	1,96	1,91	2,00	**1,05**	1,85	**0,97**
antony	1,84	1,79	1,88	**1,05**	1,69	**0,94**
much_ado	1,88	1,80	1,89	**1,05**	1,77	**0,98**
ch00	3,28	2,69	3,00	**1,12**	2,88	**1,07**
ch01	2,69	2,20	2,43	**1,10**	2,46	**1,12**
ch02	1,76	1,43	1,70	**1,19**	1,57	**1,10**
ch03	2,90	1,87	2,70	**1,44**	2,08	**1,11**
ch04	2,09	1,66	1,78	**1,07**	1,83	**1,10**
ch05	2,28	1,81	2,03	**1,12**	2,04	**1,13**
glossary	2,07	1,64	1,84	**1,12**	1,89	**1,15**
howto	6,69	2,30	2,50	**1,09**	2,59	**1,13**
hledani	3,79	3,13	3,62	**1,16**	3,40	**1,09**
komunikace	3,25	2,65	2,93	**1,11**	3,01	**1,14**
navihace	3,79	3,14	3,68	**1,17**	3,44	**1,10**
robot	3,43	2,86	3,22	**1,13**	3,04	**1,06**
xml	3,74	3,23	3,69	**1,14**	3,30	**1,02**
rur1	2,33	2,07	2,37	**1,14**	2,15	**1,04**
Average	**2,88**	**2,22**	**2,51**	**1,13**	**2,38**	**1,07**

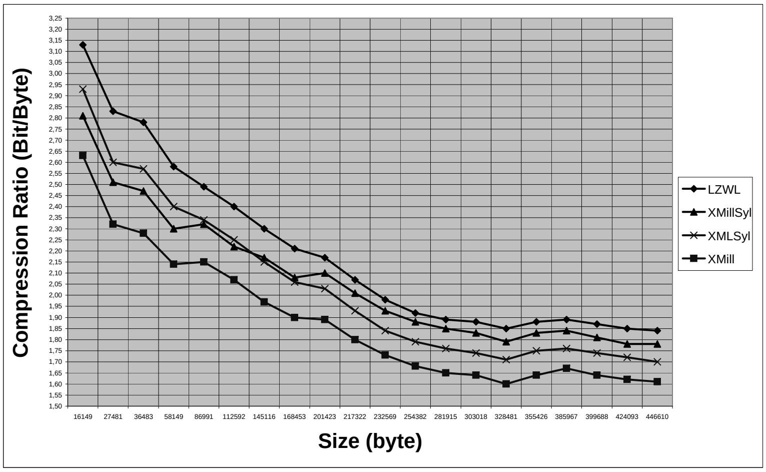

Fig. 4. Compression ratio under different sizes.

of our tools and tested their performance on a variety of XML documents. In our experiments, XMLSyl and XMillSyl were compared with LZWL and XMill. Both methods are more suitable for textual XML documents. XMill outperformed our methods only marginally. XMLSyl performs better than XMillSyl. It implies that in our case encoding of XML structure is more efficient than separating a structure from data and grouping data values with related meaning. XMillSyl and XMLSyl show better results for Czech language.

In the future, we want implement some modifications to enhance the compression ratio. For example, the information in the DTD section can be extracted and utilized to create a special syllable dictionary for elements and attributes.

References

1. Wilfred Ng, Lam Wai, Yeung James Cheng. Comparative Analysis of XML Compression Technologies. World Wide Web Journal, 2005
2. Smitha S. Nair. XML Compression Techniques: A Survey. `www.cs.uiowa.edu/~rlawrenc/research/Students/SN_04_XMLCompress.pdf`
3. J. Cheney. Compressing XML with Multiplexed Hierarchical PPM Models *In Proc. Data Compression Conference*, 2001.
4. V. Toman. Compression of XML Data. MFF UK, 2003
5. World Wide Web Consorcium. Extensive Markup Language (XML) 1.0. `http://www.w3.org/XML/`
6. P. Tolani, J. R. Haritsa. XGrind: A Query-friendly XML Compressor. *In Proc. IEEE International Conference on Data Engineering*, 2002.
7. SAX: A Simple API for XML. `http://www.saxproject.org`
8. H. Liefke, D. Suciu. XMill: an Efficient Compressor for XML Data. *In Proc. ACM SIGMOD Conference, 2000.*
9. *Jun-Ki Min, Myung-Jae Park, Chin-Wan Chung, XPRESS: A Queriable Compression for XML Data* SIGMOD 2003, June 912, 2003, San Diego, CA, 2000.
10. Expat XML Parser. `http://expat.sourceforge.net`
11. T. A. Welch. A technique for high performance data compression. *IEEE Computer, 1984.*
12. *J. Lansky, M. Zemlicka. Text Compression: Syllables.* DATESO, 2005
13. J. Lansky, Slabiková komprese. MFF UK, 2005
14. V. Toman. Komprese XML dat. `http://kocour.ms.mff.cuni.cz/~mlynkova/prg036/`
15. J. Kosek. Inteligentní podpora navigace na WWW s využitím XML. `http://www.kosek.cz/diplomka/`, 2002
16. DocBook `http://www.docbook.org/`
17. A Quick Introduction to XML. `http://www.cellml.org/tutorial/xml_guide`
18. M. Pilgrim. What Is RSS. `http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html`
19. XML Processing. `http://diveintopython.org/xml_processing/`
20. SAX And DOM Overview. `http://www.jezuk.co.uk/cgi-bin/view/arabica/SAXandDOMIntro`
21. The gzip home page. `http://www.gzip.org/`