XML Lossy Text Compression: A Preliminary Study

Angela Bonifati[1,2], Marianna Lorusso[2], and Domenica Sileo[2]

[1] Italian National Research Council (CNR)
Via P. Bucci 41C, I-87036 Rende, Italy
bonifati@icar.cnr.it
[2] Dipartimento di Matematica e Informatica, University of Basilicata
Viale dell'Ateneo Lucano 10, I-85100 Potenza, Italy
marianna.lorusso@gmail.com, domenica.sileo@gmail.com

Abstract. Lossy compression techniques have been applied to image and text compression, yielding compression factors that are vastly superior to lossless compression schemes. In this paper, we present a preliminary study on a set of lossy transformations for XML documents that preserve the semantics. Inspired by previous techniques, e.g. lossy text compression and literate programming, we apply a simple algorithm to XML syntactic constructs to loose superfluous layout information and redundant text. The obtained XML keeps the human-readability and machine-readability properties. Additionally, it can lead to a considerable reduction of its space occupancy and boost the application of conventional text compressors, thus representing a promising technology for several data management tasks.

1 Introduction

Lossy compression leads to produce compressed files that cannot be reconstructed in their original form. Such compression can be used alone or in conjunction with lossless compression to improve the compression rate. We focus on lossy compression techniques to be applied to synthetic languages and to natural language as well. An XML file is indeed a combination of both languages, as it consists of syntactic constructs, such as elements, attributes, comments, entities, processing instructions etc. and of natural text, which is embedded into PCDATA nodes. As such, XML can be shrunk by eliminating layout information, such as whitespaces and closing tags. Such techniques have been devised for natural languages in lossy text compression [1], where thesauri-based compression is employed, by replacing words with their shorter synonyms. In synthetic languages, the lossy text compression intends to compress the text but preserve the semantics. For instance, source code can be compressed by eliminating superfluous white space. The obtained program will not be human-readable as it used to be, but will still be accepted by its compiler. Not only layout information but also comments are eliminated from a source program in WEB [2], to obtain a 'tangled' version of the language, which is not intended for human consumption but only for compilers consumption. Literate programming aims at creating

Z. Bellahsène et al. (Eds.): XSym 2009, LNCS 5679, pp. 86–93, 2009.

two versions of a given program, a compiler version and a pretty-printing tex version, the latter being suitable as code documentation. Finally, the common technique of omitting vowels from text, while being hardly suitable for practical use, sacrifices readability for compression. Its variations include Speedwriting [3], Braille [4] and Soundex [5].

All the aforementioned techniques are not directly applicable to XML syntactic constructs and need to be customized. XML has a double-fold nature, in that it has instructions (i.e. the components of the data model) and it also encloses textual data. These data typically reside in the leaves and can be quite large for full-text documents. Lossy compression of natural language has been tackled in the past, by yielding outputs seldom readable by humans. Whereas these solutions were acceptable for text, they are not viable for XML data, that has to keep its human-readability and machine-readability at any rate.

Our first contribution has been that of devising a set of rules for lossy text compression of an XML file. We have identified a set of rules applicable to the syntactical constructs of an XML file, a set of rules for compressing its textual content, and a set of rules to reduce its formatting content. We obtain two variants of an XML file, that are both machine-readable, the first preserving the well-formedness property, and the second sacrificing it for compression. We call these variants $\mathcal{WF}$-Lossy XML and $\mathcal{TF}$-Lossy XML ($\mathcal{WF}$-LX and $\mathcal{TF}$-LX, in short notation). Before explaining the acronyms, we observe that, while the first of the two variants is still an XML file, thus can be processed as such, both variants do keep the human-readability property. $\mathcal{WF}$-LX files represent well-formed Lossy XML files, whereas $\mathcal{TF}$-LX represent text-formed Lossy XML files, i.e. files that can be read as text, although their conversion to a well-formed document is straightforward (cfr. Section 2). *Our second contribution has been that of implementing a Lossy XML Compressor (LXC), capable of yielding both formats, and studying its effectiveness on several XML datasets.* From our analysis, we observed that our rules let achieve a moderate compression factor in some of the considered datasets, and a significant reduction (up to 40%) in a few other datasets. *Our third contribution is that of identifying a class of applications, in order to show the utility of our approach.* It does appear that some very interesting work can be done in this area.

The rest of the paper is organized as follows. Section 2 describes the rules to obtain lossy XML files. Section 3 shows a set of experiments to gauge the effectiveness of our technique and its usage in conjunction with ordinary compressors. Section 4 discusses the related work. Finally, Section 5 concludes our paper, and discusses future directions of our work.

2 Rules for XML Lossy Text Compression

We describe in the following the rules we have devised to compress the textual content of an XML file in a lossy fashion. These rules are divided into three main categories: *(i)* PCData rules, i.e. only applicable to the leaves of an XML tree, being such leaves the PCData values of attributes or elements; *(ii)* Tag

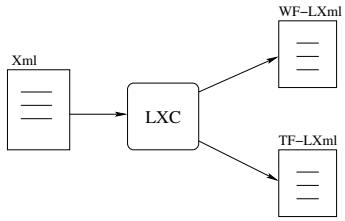

Fig. 1. Lossy XML Compression (LXC) yields two output files

name rules, i.e. only applicable to intermediate nodes of an XML tree, i.e. to the starting and closing tags of an element, to attribute names and to namespace nodes; *(iii)* Formatting rules, i.e. applicable to various formatting characters that lie in between nodes in an XML tree.

PCData Rules. The rules for PCDATA are as follows:

1. a sequence of one or more vowels is replaced with one vowel, typically the first;
2. a sequence of one or more punctuation characters is replaced with the empty sequence if there is a blank afterwards, otherwise with a whitespace; this rule is not applied in cases where the punctuation is followed by a number, or when the word begins with the character '&' and ends with the character ';' as this represents an entity;
3. a sequence of two or more formatting characters is replaced with one formatting character;
4. 's and s' (genitive inflection) are eliminated from words;
5. a word duplicate if it appears after the first duplicate is eliminated;
6. numbers written in letters are replaced with the corresponding digits;
7. whitespaces in the leading sentence of a given paragraph and whitespaces in the trailing sentence of a paragraph are deleted;
8. if there is an acronym followed by its full expansion, the latter is eliminated; any full expansion in the document is replaced by the acronym;
9. end of line characters are removed and replaced by a whitespace;
10. if a word that appears after the character '.', '!' and '?' begins with a lowercase letter, the letter is converted in uppercase.

Formatting Rules. The rules for formatting are as follows:

1. end of line characters and lines of whitespaces are eliminated;
2. comments are removed;
3. indentation characters are removed;
4. sequences of two or more formatting characters are replaced with one formatting characters, typically the first;
5. leading and trailing whitespaces are removed.

Table 1. Prioritized order among rules - table must be read *per row*; TR (PR, FR, resp.) stands for Tag Rule (PCData Rule, Formatting Rule, resp.)

FR3	FR2	*TR3*	FR5	PR8	PR4
PR5	PR10	PR2	PR6	FR4	PR3
PR1	PR7	TR2	TR1	PR9	FR1

Tag Rules. The rules for tag names, attributes and namespaces are as follows:

1. sequences of two or more vowels are replaced with one vowel, typically the first;
2. characters '. ', '-' and '_' are removed;
3. each closing tag name is eliminated, and left as the acute brackets with the character '/' (i.e. `</>`).

We have prioritized the rules both within each class and across distinct classes. This boils down to decide a global ordering for rules, which can be read per row in Table 1. To give an intuition, rules need to be prioritized in order to avoid redundant work (e.g. PCData rule 6 is applied before PCData rule 1, in order to avoid collapsing consecutive vowels in numbers) and to guarantee that the application of a rule is not void (e.g. PCData rule 10 is fired before PCData rule 2, in order to avoid loosing the separation into sentences before removing redundant punctuation characters). All rules must be applied to obtain a $\mathcal{WF}$-LX file. All rules but Tag Rule 3 are applied to obtain a $\mathcal{TF}$-LX file.

Before discussing the implementation of our prototype, we would like to underscore the importance of having Tag Rule *TR3*, which substitutes the final closing tag of an element with its empty version `</>`. If such substitution takes place, the obtained $\mathcal{TF}$-LX document looses its well-formedness. However, such document, as we will see from the experimental result, becomes more apt to compression. Notwithstanding the advantages of keeping the document in textual format, one can always reconstruct the XML closing tags and transform every $\mathcal{TF}$-LX document into an $\mathcal{WF}$-LX document. We also observe that a depth-first encoding can be applied to $\mathcal{TF}$-LX documents, similarly to that applied to n-ary trees or balanced mathematical expressions [6]. Indeed, closing empty tags act as placeholders, and can be easily matched to opening tags during document reconstruction. As such, closing tags resemble closing parentheses in mathematical expressions. An balanced parentheses encoding [6] can be applied to the obtained lossy $\mathcal{TF}$-LX document, which we plan to investigate as future work.

3 Experimental Study

We have conducted an experimental study in order to gauge the effectiveness of our technique and the succinctness of the obtained documents. In particular,

Table 2. XML documents used

Document d	Size (KB)	# Elems.	# Attributes	Max Depth	Provenance
Path	203	2764	8627	10	[7]
XMark	113,794	1,666,315	381,878	13	[8]
DBLP	932,444	16,272,139	14,936,399	7	[9]
Shakespeare	274	6636	0	8	[10]
TreeBank	84,065	2,437,666	1	37	[9]
SwissProt	112,130	2,977,031	2,189,859	6	[9]
News	238,677	3,974,681	0	3	[11]

we have run a first set of experiments, aiming at measuring the compression ratio of both $\mathcal{WF}$-LX documents and $\mathcal{TF}$-LX documents. Then, we have compared these compression factors with existing compressors. Next, we have run a second set of experiments, by sequencing the application of our lossy compression technique and of general-purpose compressors. Finally, as a third set of experiments, we have loaded our $\mathcal{WF}$-LX documents into an XML database engine and measured both the loading time and the query execution times.

3.1 Experimental Setting and Results

We have performed our experiments on a Windows XP Pro Laptop with 2.40 Ghz Intel Core Duo CPU P8600, and 4 GB RAM. The datasets used in the experimental study are shown in Table 2. Their structure varies from flat (e.g. DBLP) to deeply nested (e.g. XMark and TreeBank).

Figure 2 (top) shows the compression ratio obtained for the above datasets by using our tool LXC. We can observe that the effectiveness of the rules is higher once the textual content of a document is higher, and that the technique performs quite well in some datasets, such as DBLP and TreeBank (up to 47%). Such datasets exhibit both compressible tags and compressible textual content. The remaining datasets can achieve a moderate compression ratio, that goes from 20% to 10%. In Figure 2 (bottom), we compare our results with the compression ratios obtained by two general-purpose compressors, such as GZip and BZip2 [12], two XML compressors, i.e. XMill [13] and Xmlppm [14], and Huffword (word-oriented Huffman [15]). Although our technique is fairly different from the opaque lossless compression obtained by the above tools, we can observe that the compression ratio obtained by our lossy technique can be up to an half of the compression ratio of both classical and XML compressors, and can be comparable to the compression ratio achieved by Huffword on some datasets.

We have next analyzed the impact of our technique on the effectiveness of classical and XML compressors. Hence, we have sequentially applied our technique and classical compressors. We did not observe a significant variation of the compression ratio for GZip, BZip2, XMill and Xmlppm. Indeed, by either

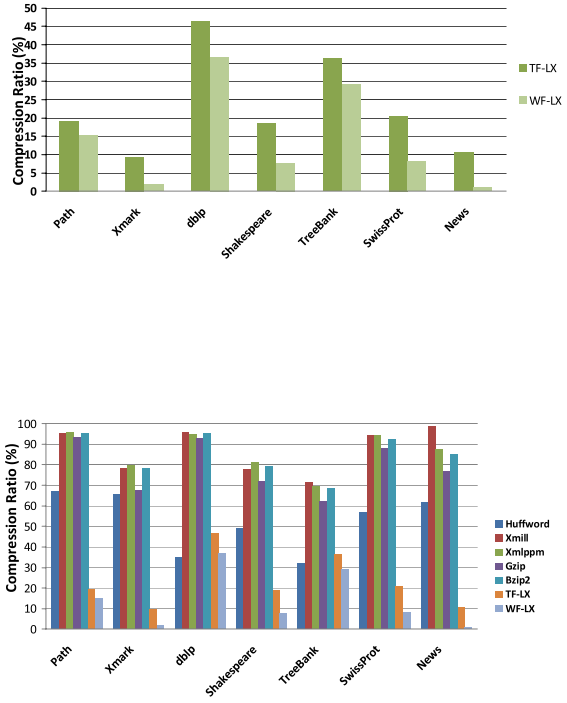

Fig. 2. Compression Ratio (%) for various XML datasets of $\mathcal{TF}$-LX against $\mathcal{WF}$-LX (top); Compression Ratio (%) for various XML datasets of $\mathcal{TF}$-LX and $\mathcal{WF}$-LX (bottom) against other compression tools

applying GZip (BZip2, resp.) to $\mathcal{TF}$-LX files[1], or by applying GZip (BZip2, Xmill, Xmlppm, resp.) to $\mathcal{WF}$-LX files, the compression ratio of these tools is the same of that achieved when working on the original document, or, in some cases, slightly decreases. This demonstrates that the above compressors either ignores the changes applied by our rules, or cannot take advantage of them. Due to the lack of space, we opt not to present the above results and only show the figures relative to Huffword. Indeed, this was the only compressor that could exploit the reduction induced by our technique. Figure 3 shows the results of applying Huffword to the $\mathcal{TF}$-LX files, Huffword to the original files, and Huffword to the $\mathcal{WF}$-LX files, by using the datasets of Table 2. We can observe that in most of the cases, lossy compression boosts Huffword compression ratio, or, at least, does not change it. Only in one case, the compression ratio decreases, and this happens with TreeBank. The motivation behind this is the fact that TreeBank contains partially encrypted data, and these data cannot be processed by our technique. This result lets us thinking that TreeBank compression ratio would have improved if those data were decrypted.

[1] We did not employ XMill and Xmlppm in this experiment, as these are not applicable to $\mathcal{TF}$-LX files.

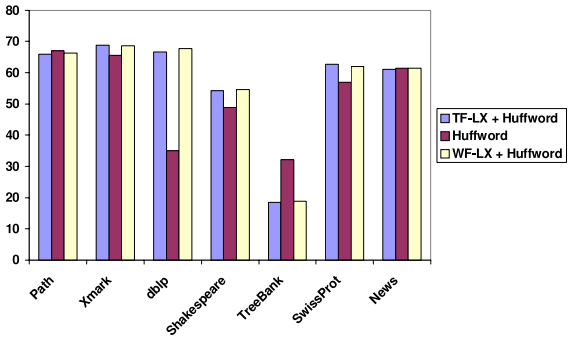

Fig. 3. Compression Ratio (%) for various XML datasets of applying $\mathcal{TF}$-LX and Huffword, Huffword alone, and $\mathcal{WF}$-LX and Huffword

To conclude our experiments, we showed the utility of $\mathcal{WF}$-LX documents into XML databases. We considered QizX [16], a quite fast XML database and XQuery engine, and we uploaded TreeBank and DBLP[2] after applying lossy compression. We observed that the loading time of $\mathcal{WF}$-LX documents was on average 10% less than the loading time of the original documents. Then, we executed a set of XPath queries on both datasets and observed a reduction of 20% of the query execution times on average.

4 Related Work

Techniques for word abbreviations have been the subject of past research on textual data [5], where various word abbreviation techniques are compared, among which Soundex, a phonetic algorithm for indexing names patented by R.C. Russel in 1918. The Soundex indexing system is still in use nowadays by the US Census, to systematically code persons' names. Each word in the Soundex alphabet retained the same degree of discrimination of the original word, along with the mnemonic similarity with the original word. Moreover, the procedure can be applied on the fly, i.e. without any prior knowledge of the population of words, and typically at any new person filed at Census. We observe that the no storage or table lookup is necessary in our technique and the above principles are as well satisfied by our rules. Whereas a large body of research studied the problem of opaque compression for XML files [13,14,17,18], to the best of our knowledge no previous work aimed at envisioning abbreviation techniques for XML constructs and textual content. The only work we are aware of about lossy compression is [19], but their aim is to build a synopsis of the original document, reporting aggregate data, which is quite different from the text abbreviation rules we have presented in this paper.

[2] We only used TreeBank and DBLP in these experiments, as they obtained the highest compression ratio under our technique.

5 Conclusions and Future Perspectives

We have described our preliminary study on XML lossy text compression. We believe that there exists several interesting directions of future research. In particular, many challenges are left to be addressed, such as designing a full-text engine for $\mathcal{TF}$-LX files and studying keyword-based queries for such documents. Another important milestone is to devise an extension to XQuery that handles queries on $\mathcal{WF}$-LX files in the compressed domain, as in our past work [17].

References

1. Witten, I.H., Bell, T.C., Moffat, A., Nevill-Manning, C.G., Smith, T.C., Thimbleby, H.W.: Semantic and generative models for lossy text compression. The Computer Journal 37(2), 83–87 (1994)
2. Knuth, D.E.: Literate programming. The Computer Journal 27(2), 97–111 (1984)
3. SpeedWriting, `http://www.speedwriting.co.uk/`
4. Braille, `http://www.nfb.org/nfb/BrailleInitiative.asp`
5. Bourne, C.P., Ford, D.F.: A study of methods for systematically abbreviating english words and names. J. ACM 8(4), 538–552 (1961)
6. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing Trees of Higher Degree. Algorithmica 43(4), 275–292 (2005)
7. PathWays, `http://www.genome.jp/kegg/xml`
8. Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., Busse, R.: XMark: A benchmark for XML data management. In: Proceedings of VLDB, pp. 974–985 (2002)
9. University of Washington's XML repository (2004), `www.cs.washington.edu/research/xmldatasets`
10. Ibiblio.org web site (2004), `www.ibiblio.org/xml/books/biblegold/examples/baseball/`
11. AG's corpus of News articles, `http://www.di.unipi.it/gulli/newsspace200.xml.bz`
12. The bzip2 and libbzip2 Official Home Page (2002), `http://sources.redhat.com/bzip2/`
13. Liefke, H., Suciu, D.: XMILL: An Efficient Compressor for XML Data. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, pp. 153–164. ACM, New York (2000)
14. Cheney, J.: Compressing XML with Multiplexed Hierarchical PPM Models. In: DCC, pp. 163–172 (2001)
15. Huffman, D.A.: A Method for Construction of Minimum-Redundancy Codes. In: Proc. of the IRE, pp. 1098–1101 (1952)
16. Qizx, `http://www.xfra.net/qizxopen/`
17. Arion, A., Bonifati, A., Manolescu, I., Pugliese, A.: XQueC: A query-conscious compressed XML database. ACM Trans. Internet Techn. 7(2) (2007)
18. Ng, W., Lam, Y.W., Cheng, J.: Comparative Analysis of XML Compression Technologies. World Wide Web Journal 9(1), 5–33 (2006)
19. Cannataro, M., Carelli, G., Pugliese, A., Saccá, D.: Semantic Lossy Compression of XML Data. In: Proceedings of KRDB (2001)