Semantic Lossy Compression of XML Data

Mario Cannataro[2], Gianluca Carelli[1], Andrea Pugliese[1,2], Domenico Saccà[1,2]
D.E.I.S. Department, University of Calabria, Via P. Bucci, Rende, Italy
ISI-CNR, Via P.Bucci, Rende, Italy
{cannataro, carelli, pugliese, sacca}@isi.cs.cnr.it

1 Introduction

In the last years a large amount of *semistructured* data [1, 10] has been managed and exchanged. The largest repository of semistructured data is the World Wide Web, which can be thought of as an enormous database in which data is highly heterogeneous and freely correlated. In this scenario is placed *Extensible Markup Language* (*XML*) [14], a language for semistructured data standardised by the *World Wide Web Consortium* (*W3C*), which is candidate to become shortly the *de facto* standard for web documents. XML allows building *machine-readable* documents that are naturally convertible in visualisation formats; this is obtained by means of a complete separation among structure, content and style of documents.

It is likely that the amount of data available in XML will grow substantially, e.g. in those applications in which the generation of XML documents is performed automatically from data maintained in DBMS. The increasing amount of XML data will lead to the origin of new issues regarding efficiency in the representation of documents. An emerging problem is how to compress the description of an XML document. An interesting solution is *XMill* [7] that is a lossless (i.e., the original data are eventually restored) compressor/decompressor for XML data, to be used in data exchange and archiving. XMill applies classical entropy-based compression techniques after the execution of ad-hoc compression rules that are driven by the XML structure of the document and by the semantics of data. Its main ideas are *(i)* separating structure from data, *(ii)* grouping related data items into homogeneous classes, *(iii)* applying semantic compressors to those data classes and *(iv)* applying general-purpose compressors.

XMill is a very effective compression tool which overpasses classical general-purpose compressors such as `gzip` [3]. The problem with XMill is that, as for classical text compressors, the compression is only used for archiving or exchanging but not for deriving a "synthetic" yet meaningful view of a document as it happens in the compression of images (JPEG) or video sequences (MPEG). Our belief is that lossy compression will become relevant in next applications on Internet.

The typical scenario we are envisioning is a multi-channel access to XML documents whereby a document may be required to be displayed e.g. on a small-sized screen using a low bandwidth network. In this case the admissible compression rate

could not be reached by any lossless compression so that lossy techniques must take over which remove details and provide suitable aggregations. For instance, assume that an XML document contains the day sales: each sale is characterised by the item, the store and the quantity. Say that there are 100 sales per day. The account manager may inquire the document from his/her nice 19-inch workstation over a gigabit LAN and, then, he/she will receive the document as it is. But if he/she is now at the airport issuing the query on a WAP phone, the document needs a lot of compression to be readable. The simplest solution is to prepare a document that only contains the total amount of day sales. But this is a too little piece of information. The nice, expensive WAP phone and the available network bandwidth may effort to receive additional data.

The solution we propose is that a lossy compression rate is first negotiated and then the document is delivered with this compression rate. For instance as the various sold items form a sequence, we regard them as a sort of relation; then we single out a number of dimensions (item type, customer city), eventually a hierarchy over them (city, province, state, country) and a measure (the quantity), thus providing a multidimensional representation that will finally be structured as a datacube with aggregate data on suitable dimension intervals. The resulting compressed XML document is another document, indeed a synthetic view of the original one. Moreover, we apply (lossy) semantic compression/decompression techniques to such synthetic version of the XML document, thus extending the ideas used in XMill and obtaining a further increase of performance.

A prototype of the system has been implemented. It uses an XML query processor to accomplish the structural compression, that takes as input the multidimensional description expressed in an XML-based language resembling the descriptions of classical *star schemas*. We also developed some lossless and lossy semantic compressors and an XML-based language for describing possible associations between homogeneous parts of documents and semantic compressors.

The rest of the paper is organised as follows: in Section 2 specific XML characteristics are explained, which can be used to improve compression effectiveness; in Section 3 a system for XML compression is proposed; in Section 4 some preliminary experimental results are shown; Section 5 outlines conclusions and future work.

2 Background on XML Compression

In general, data compression is concerned with the minimisation of the amount of data needed to represent some information, which is produced by a *source* and described by *messages* composed of *symbols*. Compression is often referred to as *coding*, since its objective is to represent source messages with corresponding codes. *Source* coding is related with the semantics of data, whereas *entropy* coding refers only to its redundancy [2, 5, 8].

The most important distinction among compression techniques concerns with their *reversibility*. If decoded data are identical to original ones, compression is called *lossless*; otherwise, it is called *lossy*. Lossless compression schemes refer e.g. to the work

by Huffman [4, 5], the algorithm by Lempel and Ziv [6] and the more recent *Arithmetic Coding*[5]. Lossy compression allows to obtain significantly higher compression ratios preserving a representative subset of original data; interesting approaches to lossy compression are *wavelet transformations* [11], *histograms* [9, 12] and methods for the extraction of significant parts from a free text [13].

XML uses markups to identify and describe data; the schema-related information is contained in documents themselves so the language is called *self-describing*. Therefore, it is always possible to directly identify data through their *paths*; this feature becomes specially useful when when data types are strictly constrained (e.g. for documents built from data stored in DBMSs), as it becomes possible to associate particular paths to specific data types.

On the other hand, these aspects lead to verbosity and modest efficiency since markup structure is clearly not economic (terseness is considered "of minimal importance" also in language specification [14]); tags exhibit noticeable redundancy, closing tags must contain elements' names and different tags can be repeated many times in the documents, each time in an extended form. In [7] are shown examples of transformations from more space-efficient formats to XML; the gain in flexibility is balanced by an increase of documents' size (in those cases documents grow up to the 350% of the original ones).

The above-mentioned characteristics of XML are fundamental for documents compression. Regarding the markup structure, a more efficient representation of symbols can lead to a dramatic decrease of the physical space needed to store it. Furthermore, as documents are self-describing, it is possible to identify data on the basis of their type and of their semantics. Therefore, more intelligent interpretations of data can be applied (e.g. the multidimensional view presented in this paper), and compression loss can be introduced.

Recently, two interesting systems for lossless XML compression have been developed. XMill [7], as said before, makes use of the above-shown characteristics of data identifiability and markup structure compressibility, to obtain compression ratios which are significantly higher than those produced by general-purpose compressors, at almost the same speed. *XMLZip* [15] cuts the XML tree at a certain depth and compresses the two parts separately.

3 A System for the Semantic Lossy Compression of XML Documents

The main idea of our semantic lossy compressor is to process the XML document (both data and structure), in such a way elements can be regarded as tuples of a relation, to single out a number of dimensions and measures and provide a multidimensional representation that will finally be structured as a datacube, with aggregate data on suitable dimension intervals. So, the document is reorganised according to some aggregation functions, resulting in a synthetic version of the original one.

Moreover, we apply a similar approach as in XMill to further compress the XML markup structure and data. In particular, we use lossless compression techniques

for the markup structure and both lossy and lossless techniques for the data. The compressor identifies data contained in documents, groups them in specific containers and selectively compresses them, eventually introducing a certain degree of loss, thus obtaining a compression improvement. The logical architecture of the system is shown in Fig.1.

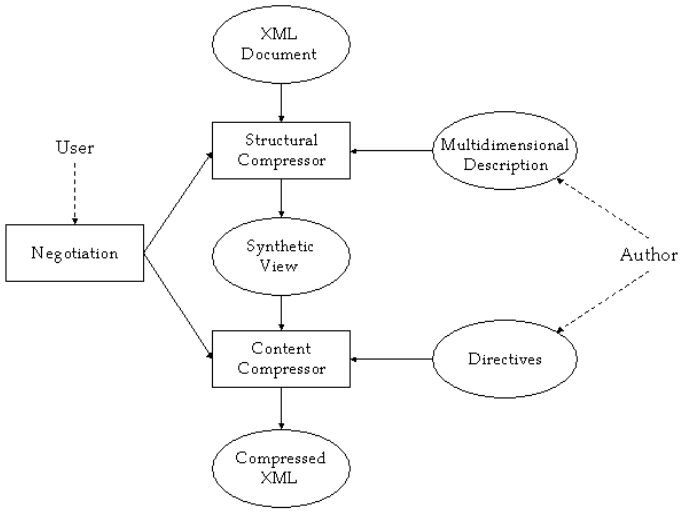

Figure 1: System Architecture.

3.1 Structural Compression

The original XML document is first synthesised through the *Structural Compressor*. It uses the *Multidimensional Description* of the document given by the author in an XML document to interpret data as a multidimensional datacube. The *detail level* (aggregation functions, dimensions of interest, hierarchies over them etc.) is set up by means of the *Loss Negotiation* module.

For instance, consider an XML document containing elements describing sales and having the following structure:

```
<sale ID="...">
  <item>...</item>
  <time>...</time>
  <location>...</location>
  <units>...</units>
</sale>
```

Possible dimension attributes could be the item, the time and the location, and a measure attribute the units sold. A suitable hierarchy over dimensions could be the following:

$$item : item \longrightarrow type$$
$$time : day \longrightarrow month \longrightarrow quarter \longrightarrow year$$

$location : store \longrightarrow city \longrightarrow province \longrightarrow country$

The XML multidimensional description could have the following structure:

```
<dimension
  path="/sale/item"
  name="item">
  <row>
    <item>...</item>
    <type>...</type>
  </row>
  ...
</dimension>
<dimension path="/sale/time"
  name="time">
  <row>
    <day>...</day>
    <month>...</month>
    <quarter>...</quarter>
    <year>...</year>
  </row>
  ...
</dimension>
...
<measure path="/sale/units"
  name="units"
  functions="sum avg min max"/>
...
```

Here, the author *(i)* specifies that the dimension "item" is identified by the path `/sale/item`, *(ii)* specifies that the dimension "time" is identified by the path `/sale/time`, *(iii)* gives the dimension tables for them, *(iv)* specifies that the measure "units" is identified by the path `/sale/units` and *(v)* indicates a number of applicable aggregation functions.

A detail level could be set up by specifying e.g. {*item*, *month*, avg(*units*)} meaning that the user requires the average value of the *units* measure attribute, grouping data by *item* and *month*. In this case, the synthetic version of the document (Fig.2) would have the following structure:

```
<sales>
  <item>...</item>
  <month>...</month>
  <avg-units>...</avg-units>
</sales>
...
```

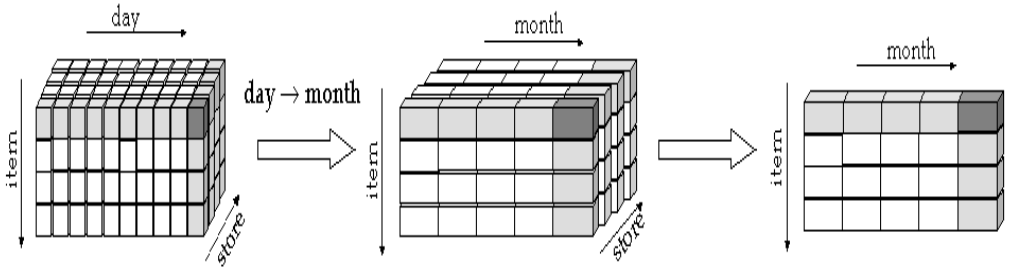

Figure 2: Graphical representation of a document synthesisation.

3.2 Content Compression

The *Parser* module (Fig.3) analyses in a serial way the XML documents. In particular:

1. It partitions the documents' content in *Prologue*, *Structure* and *Data*. The Prologue is composed by the XML declaration (version, encoding etc.), the document type declaration (name of the *Document Type Definition*, internal subset etc.) and a code table, i.e. a list of code-name associations useful for the subsequent decoding of tag names. The markup Structure, represented by means of a sequence of symbols (i.e. integer values), comprises special codes to recover data contained in documents, the set of start- and end-tags, and markups for special sections (unparsed sections, entity references, processing instructions, comments etc.). Data are attribute values, contents of elements, unparsed sections, comments etc.

2. It partitions data in a set of *Containers*, obtaining homogeneous groups (i.e. groups of data of the same type and eventually the same semantics). For the mapping from data to containers it is aided by the *Path Processor*.

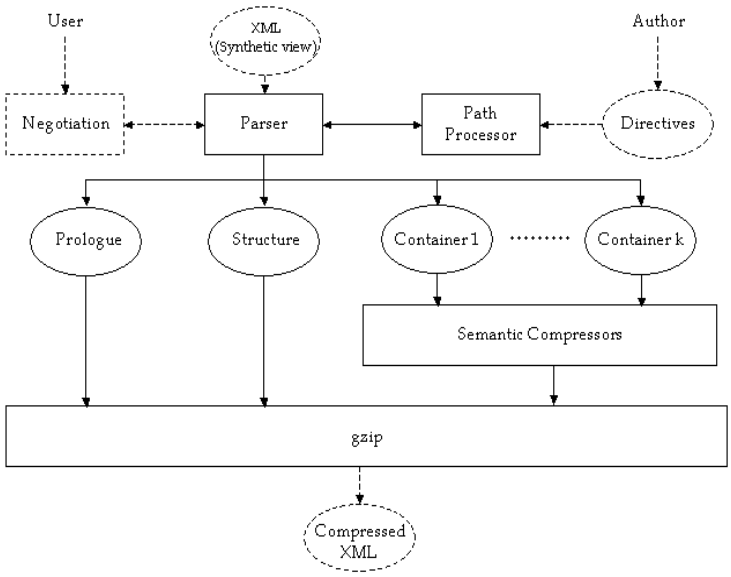

Figure 3: The Content Compressor.

The Data Containers are compressed by some *Semantic Compressors*, which perform a type- and semantic-dependent compression and eventually introduce a certain degree of loss. Thus, at the input of the `gzip` compressor the XML document is split in different parts which are essentially homogeneous, and are compressed separately. `gzip` combines Huffman coding and Lempel-Ziv algorithm [10] and therefore it is much more effective on homogeneous data.

Data contained in the document are distributed in the containers on the basis of the user's choices, expressed in the negotiation phase. The author's *directives*, contained in a valid XML document, contain some expressions regarding data paths (*Path Expressions*) and possible association criteria from paths to compressors. Path

expressions follow a particular syntax similar to the one presented in [7]. The position is specified by navigating through the document (e.g. `/root/element/sub-element`); `@` selects single attributes (`/element@attribute`); `//` indicates a descendant at any level (`/root//descendant`); the wildcards `*` and `#` respectively indicate to group data in a single container or create a new one for each different path (`//element/*` or `//element/#`).

Each possible association between path expressions and compressor is described as follows:

```
<association>
  <pattern>...</pattern>
  <semantic-compressor>...</semantic-compressor>
</association>
```

3.2.1 Semantic Compressors

As in XMill, different *ad hoc* compressors handle specific semantics of data. A fundamental aspect of our proposal is the possibility to introduce a certain degree of loss, gaining in compression performance. Moreover, we plan to allow the user developing and using "personalised" semantic compressors, eventually specialising pre-defined ones. So far we have implemented the following semantic compressors:

- *Lossless compressor for integers, reals and strings.* It uses the minimum number of bytes needed (e.g. one byte for integers in the interval [-128,+127]). The string compressor uses a different number of bytes (1, 2 or 3) to code each character of the strings.

- *Lossless compressor for IP addresses*: it simply encodes IP addresses with 4 bytes, while their *UTF-8* representation needs 15 bytes.

- *Lossy differential compressor.* It encodes exact values with a certain frequency, and represents intermediate data as a difference between adjacent values. On the basis of the number of bytes used to represent differences, a loss can be introduced. Obviously, such compressor is useful for regular data sequences.

- *Wavelet-based lossy compressor for sequences of numbers.* It uses wavelets which allow to compress a sequence of numbers in an effective way and with a precise control of introduced errors. This approach is particularly useful for compressing XML documents with huge sequences of numbers, e.g. documents containing scientific or financial data. The compressor implements a simple wavelet technique proposed by Haar [11].

- *Lossy compressor for sequences of records.* It replaces measure values with ranges, and represents such values with fewer bits.

- *Lossy compressor for strings.* It truncates strings, so it is useful when it is significant to preserve only an initial part of the strings contained in XML documents. The compressor truncates the strings on the basis of a percentage specified by the user.

- *Lossy compressor for free text.* It transforms the original text in a new one, more compact and whose meaning can be recognised by a human. The main idea is to set up in advance a list of words to be eliminated; as an example, in Italian language it could be suitable to remove articles, conjunctions etc.

4 Experimental Results

Some preliminary experimental results of the structural compression are shown in Table 1. They have been obtained on an XML input document having the structure of the example of Section 3.1, comprising 5000 records and having a size of 656770 Bytes. The Compression Factor is defined as $CF = 1 - (Compressed\,Size)/(Original\,Size)$.

Detail level	Number of elements	Compressed Size (Bytes)	Compression Factor
{*item*, *day*, *store*, *units*}	5000	656770	0%
{*item*, *quarter*, *store*, sum(*units*)}	1152	140681	78.58%
{*type*, *quarter*, *province*, sum(*units*)}	96	12056	$\simeq$ 98.1%
{*item*, *country*, sum(*units*)}	12	1239	$\simeq$ 99.7%
{*type*, sum(*units*)}	2	187	$\simeq$ 99.8%
{sum(*units*)}	1	99	$\simeq$ 99.9%

Table 1: Experimental results of the structural compression.

Furthermore, a test of the content compression was performed on a different XML document containing records with 4 *integer* fields, 5 *string* fields and one *real* field. Three different kinds of compression were used: *(case 1)* lossless, *(case 2)* applying the wavelet compressor on the real field and *(case 3)* applying the wavelet compressor on the real field, the compressor for sequences of records on one integer field and the lossy compressor for strings on one string field. The wavelet compressor was set to obtain a 2-norm average error of 5%:

$$e(V, \tilde{V}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\Big(\frac{|v_i - \tilde{v}_i|}{max(1, v_i)}\Big)^2} = 5\%$$

where $V = [v_i, \ldots, v_N]$ is the original dataset and $\tilde{V} = [\tilde{v}_i, \ldots, \tilde{v}_N]$ is the reconstructed one. The results of the test, compared with `gzip` and XMill, are shown in Fig.4. Our compressor obtains a good compression factor (about the same as XMill) if used without loss; an higher compression factor is achieved introducing different degrees of loss (cases 2 and 3).

It should be noted that in the general case, the user in the negotiation phase could decide both a level of synthesisation on the structure and a degree of loss on contents.

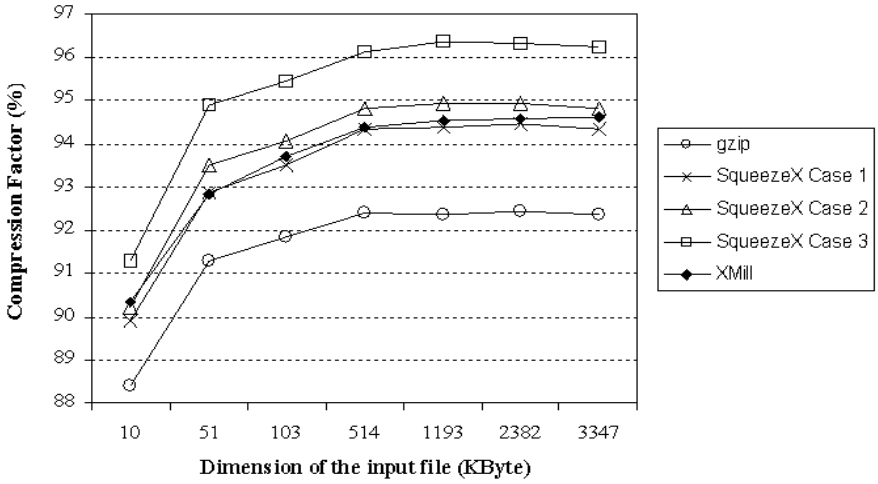

Figure 4: Experimental results of the content compression.

5 Conclusions and Future Work

In this paper we presented a lossy compression system for XML documents. The main contribution of the paper is the attempt to produce a synthetic yet meaningful version of an XML document by interpreting it as a datacube, using a multidimensional approach. The resulting reorganised XML document is further compressed using an approach similar to XMill, enhanced by the use of lossy compressors. Obviously, document restructuring makes sense in particular when *database-like* rather than *narrative* XML documents are considered.

Future work will concern an enhanced analysis of the compression quality, in terms of error. Many lossy compressors yield measurable errors; in such cases, errors can be evaluated by considering the differences between original and reconstructed data or, as for wavelet transform, they can be set-up prior to compress data. In the structural compression, where documents are interpreted as multidimensional datacubes, original detailed data could be estimated using suitable interpolation; in [9] it is also shown how to compute the errors of such estimations, in terms of expected value and variance.

Furthermore, the possibility of interpreting an XML document as a datacube could prove useful in several applications; the compressor could be used as an author system for the design of Web sites (which eventually integrate data from different Web sources) to be browsed at different detail levels.

Finally, we will consider the possibility of extending the system with the introduction of a semi-automatic knowledge extraction phase, comprising *(i)* the discover of frequent similar element structure, and feasible attributes to be considered as dimensions and measures and *(ii)* the construction of *Semantic Search Engines* that make use of our multidimensional approach for the extraction of synthetic views.

References

[1] Abiteboul, S., "Querying semi-structured data", in *Proceedings of ICDT*, 1997.

[2] Bell, T., Witten I.H., Cleary J.G., "Modeling for text Compression", *ACM Computing Surveys* 21,4 (Dec.) : 557-591, 1989.

[3] Deutsch, P., "`gzip` file format specification versione 4.3", RFC 1952, 1996.

[4] Knuth, D.E., "Dynamic Huffman Coding", *Journal of Algorithms* 6,2, 1985.

[5] Lelewer, D.A., Hirschberg D.S., "Data compression", *ACM Computing Surveys* 19,3 (Sept.): 261-266, 1987.

[6] Lempel, A., Ziv J., "A Universal Algorithm for Sequential Data Compression", *IEEE Transaction on Information Theory* 23,3 (May) : 337-343, 1977.

[7] Liefke, H., Suciu D., "XMill: an efficient compressor for XML data", *Proceedings of SIGMOD Conference*, 2000.

[8] Roth, M.A., Van Horn S.J., "Database Compression", *SIGMOD Record* 22(3) : 31-39, 1993.

[9] Saccà, D., Buccafurri F., Furfaro F., "Estimating range queries using aggregate data with integrity constraints: a probabilistic approach", in *Proceedings of ICDT*, 2001.

[10] Suciu, D., "Semistructured data and XML", in *Proceedings of International Conference of Foundations of Data Organization*, 1998.

[11] Sweldens, W., Schroder P., "Building your own wavelets at home", in *Wavelets in Computer Graphics*, ACM SIGGRAPH Course Notes, ACM Press, 1996.

[12] Vitter, J.S., Wang M., Iyer B.R., "Data Cube Approximation and Histograms via Wavelets". In *Proceedings of the 1998 ACM CIKM*, 1998.

[13] Witten, I.H., et al., "Semantic and generative models for lossy text compression", *The Computer Journal*, Volume 37, Issue 2, 1994.

[14] World Wide Web Consortium, "Extensible Markup Language", Recommendation, 2000.

[15] XML Solutions, *XMLZip*, http://www.xmlzip.com.