Combining Structural and Textual Contexts for Compressing Semistructured Databases [*]

Joaquín Adiego[1], Pablo de la Fuente[1] and Gonzalo Navarro[2]

[1]Dpto. de Informática, Universidad de Valladolid, Valladolid, España.
{jadiego, pfuente}@infor.uva.es
[2]Dpto. de Ciencias de Computación, Universidad de Chile, Santiago, Chile.
gnavarro@dcc.uchile.cl

Abstract

We describe a compression technique for semistructured documents, called SCMPPM, *which combines the Prediction by Partial Matching technique with Structural Contexts Model (SCM) technique. SCMPPM takes advantage of the context information usually implicit in the structure of the text. The idea is to use a separate PPM model to compress the text that lies inside each different structure type (e.g., different XML tag). The intuition is that the distribution of the texts that belong to a given structure type should be similar, and different from that of other structure types. This should allow PPM to make better predictions. We test our idea against plain PPM modelling, as well as against other structure-aware techniques. Results show that the new compression method obtains significant improvements in compression ratios.*

Keywords: *PPM, Compression Model, Semistructured Documents.*

1 Introduction and Related Work

Our goal in this paper is to explore the possibility of considering the text structure in the context of compressed structured documents. Structure has semantic meaning, but classical compressors do not profit from it. We aim at taking advantage of such structure. Although this idea is not new, very few compression methods are based on it

A compression method that considers the document structure is *XMill* [LS00], developed in AT&T Labs. *XMill* is an XML-specific compressor designed to exchange and store XML documents. It is based on the *zlib* library, which combines Ziv-Lempel compression [ZL77] with a variant of Huffman [Huf52].

Another approach is *XMLPPM* [Che01], an adaptive PPM-based compressor where the context given by the path in the structure tree is used to model the text in the subtree. That is, different models are used to code tag names, attribute names, attribute values, textual content, and so on. *XMLPPM* is based on the intuition that the text under similar parts should follow a similar distribution.

SCM [ANdlF03] is a generic model to compress semistructured documents, which takes advantage of the context information usually implicit in the structure of the text. The idea is that the vocabulary distribution of all the texts that belong to a given structure type should be similar, and different from that of other structure types. For example, in an email archive, in that each message is represented like a semistructured document, a different model would be used for each of the fields `<From>`, `<Subject>`, `<Date>`, `<Body>`, and so on.

SCM concept was tested using a word-based Huffman coding, which is the standard for compressing large natural language textual databases. The compression method obtained significant improvements in compression ratios. It was also shown that storing separate models may not pay off if the distribution of different structure types is not different enough, and a heuristic to *merge* models was presented with the aim of minimizing the total size of the compressed database.

In this paper we apply the SCM concept in a different way. We use separate models to compress the text that lies inside different tags. Instead of Huffman we use PPM-based models for each structural element, and an arithmetic coder.

Our experimental results show significant gains over the methods that are insensitive to the structure (including plain

[*]This work was partially supported by CYTED VII.19 RIBIDI project (all authors), TIC2003-09268 project, MCyT, España (first and second authors) and Millennium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile (third author).

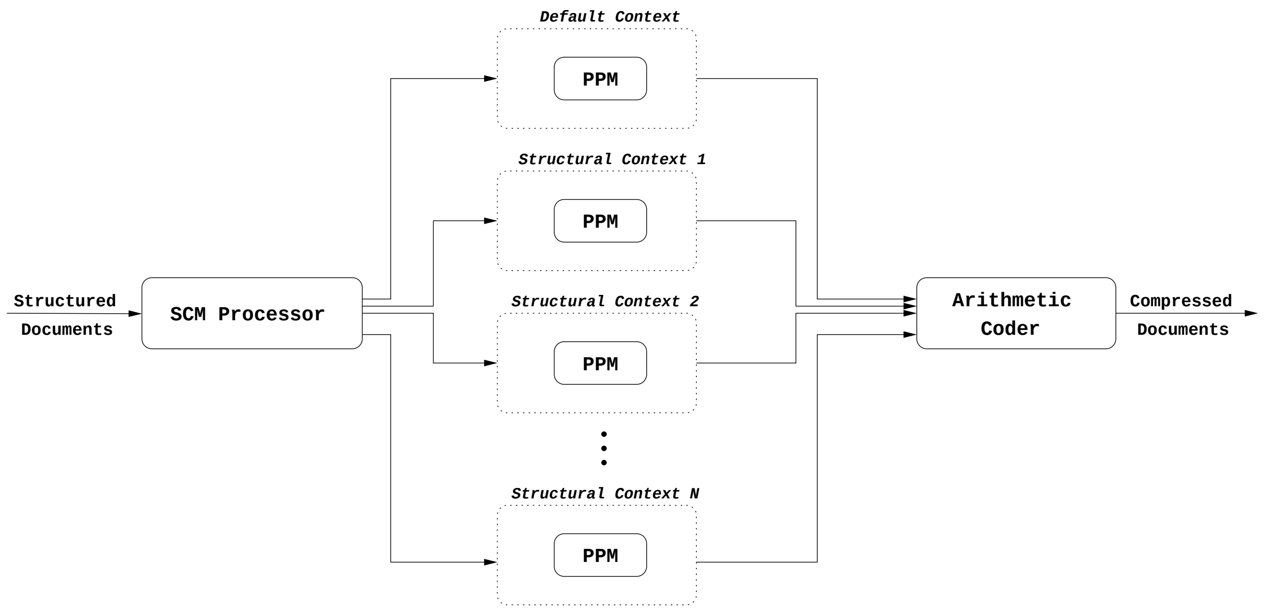

Figure 1. SCMPPM conceptual block diagram

PPM), as well as over the other methods that consider the structure.

2 Prediction by Partial Matching

We must briefly review the Prediction by Partial Matching (PPM) data compression scheme. PPM is a finite-context statistical modelling technique that can be viewed as blending together several fixed-order models to predict the next character in the input sequence. Models that condition their predictions on a few immediately preceding symbols are called finite-context models of order k, where k is the number of preceding symbols used. PPM uses a suite of fixed-order context models with different values of k, from 0 up to some pre-determined maximum, to predict upcoming characters. Probability estimation works as follows: if the symbol has been seen in the longest matching context, then the probability is the relative frequency in the context and the symbol that actually occurs is encoded relative to its predicted distribution using arithmetic coding. Otherwise, an *escape symbol* is encoded, and the next longest context is tried, and so on. The decoder maintains the same model and uses symbols seen so far and escape symbols to decode incoming symbols and update its model.

3 SCMPPM

Structural Contexts Model (SCM) is a generic compression model based on the idea that texts under same tags should have similar distributions [ANdlF03]. In most cases, natural language texts are structured in a semantically meaningful manner. This means that we can expect that, at least for some tags, the distribution of the text that appears inside a given tag differs from that of another tag, and consequently, if a PPM model is used to model text inside a specific tag will obtain better predictions. In our example of Section 1, where the tags correspond to the fields of an email archive, we can expect that the `<From>` field contains names and email addresses, the `<Date>` field contains dates, and the `<Subject>` and `<Body>` fields contain free text.

In cases where the text distribution under different tags is very different, the use of separate PPM models to encode the different tags contents is likely to improve the compression ratio. On the other hand, there is an additional cost when several escape characters are emitted to model new symbols repeated in the PPM models of different tags. Moreover, if we use a single PPM model instead, this single model may be of higher order using the same amount of memory.

In the SCM technique, there must exist one model called *default model*, which is the one in use at the beginning of the encoding/decoding process. Also, it is possible to have a different model for each tag, or in general we can have any grouping of tags under models. In this paper we assume that each tag has its own model and that the default is used for the text that is not under any tag.

We will obtain all the words that constitute the documents one by one. In this case, a *word* is any maximal sequence of alphanumeric or of non-alphanumeric characters. We will take into account a special case of words: *tags*. A tag is a code embedded in the text which represents the structure, format or style of the data. A tag is recognized from surrounding text by the use of delimiter characters. A common delimiter character for an XML o SGML tag are the symbols `'<'` and `'>'`. Usually two types of tags exist: *start-tags*, which are the first part of container element, `'<...>'`; and *end-tags*, which are the markup that ends a container element, `'</...>'`.

On the other hand, PPM models only work with symbols and, consequently, whenever we must code a word we must code all the symbols that form it sequentially.

At the begining of the process, the default model is used to predict the symbol probabilities. When a start-structure tag appears, we push the current model in a stack and switch to the appropriate model. When an end-structure tag is found we must return to the previous model stored in the stack. Both start-structure and end-structure tags are coded using the current model and then we switch models. The encoding and decoding processes use the same model switching technique.

The following code describes the model switching used for coding and decoding.

Algorithm 1 (Model Switching)

```
current_model ← default_model
while there are more words do
    $word$ ← get_word()
    $code/decode$(each_symbol($word$), current_model)
    if ($word$ is a start-structure tag)
      then $push$(current_model)
           current_model ← model($word$)
      else if ($word$ is an end-structure tag)
           then current_model ← $pop$()
```

Figure 1 shows a SCMPPM conceptual block diagram architecture. Structured documents are processed by the "SCM Processor" which is in charge to direct symbols to its corresponding structural context. Each symbol is modelled in a structural context by a PPM model that provides probability values, and then coded by an arithmetic coder to form the compressed file.

4 Evaluation of the Model

We implemented a prototype of SCMPPM, and used it to empirically analyze our idea and evaluate its performance. We have chosen the PPMD+ model variant for our implementation [CW84].

For the experiments we selected different size collections of WSJ, ZIFF and AP, from TREC-3 [Har95]. We concatenated files so as to obtain approximately similar subcollection sizes from the three collections, so the size in Mbytes is approximate. TREC collections follow the SGML standard.

The structuring of the collections is similar: they have only one level of structure, with the tag `<DOC>` indicating documents, and inside each document, tags indicating document identifier, date, title, author, source, content, keywords, etc. We assume that each structural tag will use a separate PPMD+ model to compress the text that lies inside it.

In Table 1 we show the compression ratios obtained for increasing subcollections of the different text collections, for SCMPPM using contexts of order 3, 5 and 7. It can be seen that compression improves with the subcollection size and with k.

Size (Mb.)	TREC-WSJ	TREC-ZIFF	TREC-AP
$k = 3$ PPM context models			
1	29.00%	25.95%	29.92%
5	28.00%	26.72%	28.83%
10	27.77%	26.88%	28.59%
20	27.66%	26.86%	28.40%
40	27.96%	26.78%	28.36%
60	27.34%	26.72%	28.32%
100	27.46%	26.78%	28.35%
$k = 5$ PPM context models			
1	26.03%	22.67%	25.97%
5	23.32%	21.85%	23.19%
10	22.46%	21.41%	22.44%
20	21.72%	21.23%	21.72%
40	21.22%	20.74%	21.29%
60	20.97%	20.46%	21.09%
100	20.56%	20.32%	20.92%
$k = 7$ PPM context models			
1	25.74%	22.31%	26.34%
5	22.98%	21.57%	23.21%
10	22.00%	21.25%	22.26%
20	21.19%	20.75%	21.28%
40	20.63%	20.09%	20.58%
60	20.00%	19.70%	20.23%
100	19.41%	19.42%	19.88%

Table 1. Sizes and compression ratios for the different collections and contexts numbers

In Figure 2 the evolution of the mean compression ratio of three collections can be observed when the number of contexts of each model and the collection size grows. The graphical representation for collections AP and ZIFF is very similar.

In Table 2 we can see a comparison of the compression performance of our SCMPPM technique (with orders $k = 5$ and $k = 7$) against the base technique, is a plain PPMD+

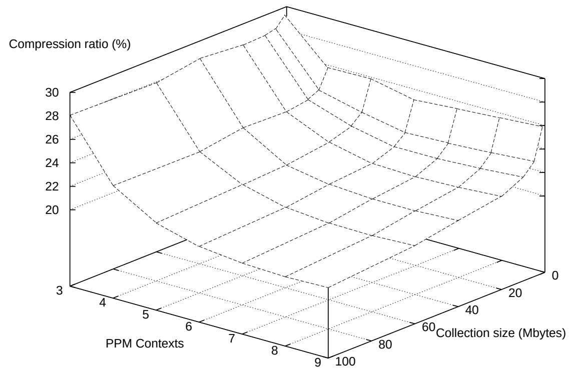

Figure 2. Evolution of the compression ratio when number of contexts and collection size grows, for TREC-WSJ.

modeler for all the text. Since it is just one PPM model, we use order $k = 7$ for it in order to use approximately the same amount of memory as our SCMPPM with $k = 5$. In this case, SCMPPM obtains improvements of up to 1%, which increases as the collection size grows. In the other hand, when we use SCMPPM with $k = 7$ (same order than base technique that using more memory) the improvement in compression is up to 6%. These results depend on the characteristics of source collections. In this case, TREC collections have a set of fields with shorts texts (dates, references, etc.) and only one a field with long free text. Therefore, the model that code this field must code almost a many symbols as the model used in the base technique.

Finally, we compare our prototype (with order $k = 5$) against other compression systems. We consider compressors that do not consider the structure and three structure-aware compressors. In the first type, we use the MG system [WMB99] and standard systems. MG system is a classic public domain software, versatile, standard and of general purpose, which handles text and images. MG compresses structured documents by handling tags as words, and uses a variant of word-based Huffman compression called *Huffword*.

Standard systems used to compare against SCMPPM are (1)*zip* and (2)*gzip*, using LZ77 plus a variant of Huffman algorithm; (3)*UNIX's compress*, that implements LZW algorithm; (4)*bzip2*, which uses the Burrows-Wheeler block sorting text compression algorithm, plus Huffman coding. *Bzip2* compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and it approaches the performance of the PPM family of statistical compressors.

The other three compressors are specific to XML: (5)*XMill* [LS00] based on Ziv-Lempel and Huffman; (6)*XMLPPM* [Che01] based on adaptive PPMD+ (with $k = 5$ order context models) over the structural context; and (7)*SCMHuff* [ANdlF03] based on the Structural Contexts Model (SCM) concept using a word-based Huffman coder.

We compressed all the collections with all systems[1] and averaged compression ratios for each collection size. Average compression ratios are shown in Table 3.

In Table 3 and Figure 3 it can be observed that the standard compressors obtain approximately constant compression ratios, except *bzip2* that applies a Huffman-type coding over a text block previously transformed by the Burrows-Wheeler Transformation, this yields a compression ratio close to PPM. SCMPPM improves the compression ratio of *bzip2* by 28%.

Word-based compressors have a convergent compression ratio when collection grows, due to the vocabulary overhead in small collections. Character-based compressors have a roughly constant compression ratio. On the other hand, our SCMPPM has a convergent ratio due to the structure overhead in small collections.

XMill obtains an average compression ratio roughly constant in all cases because it uses *zlib* as its main compression machinery. The compression ratio obtained is not competitive in this experiment: SCMPPM improves its compression ratio by 77%.

MG and *SCMHuff* improve for larger collections, as expected for word-based Huffman methods, but they stay well above the compression that can be achieved by SCMPPM, which improves them by more than 35%.

XMLPPM is the most competitive alternative to SCMPPM. Hence, although for 1 Mbyte the compression ratios are similar, for 100 Mbytes SCMPPM wins by 25%. In order to test *XMLPPM* with at least the same amount of memory that SCMPPM with $k = 5$, we modify the *XMLPPM* implementation swiching value $k = 5$ by $k = 7$ in its PPM models. This change did not produce a significant variation in the compression ratio ($\pm$0.5%), very similar to the original version. Although the ratio got slightly worse in the small collections (1 and 5 Mbytes) and improved slightly in the large ones.

5 Conclusions and Future Work

We have proposed a new method for compressing semistructured documents, combining the SCM general concept with PPMD+ modelling. We have shown that the method actually improves compression ratios by more than 1% with respect to plain PPMD+ using the same account

[1]XMLPPM required several changes to the sources in order to run properly, but these did not affect the compressibility of the collection.

Size (Mb.)	SCMPPM $k = 5$	SCMPPM $k = 7$	PPMD+ $k = 7$
1	24.89%	24.79%	24.44%
5	22.78%	22.72%	22.65%
10	22.10%	21.83%	22.12%
20	21.55%	21.07%	21.61%
40	21.08%	20.43%	21.20%
60	20.84%	19.97%	20.99%
100	20.60%	19.57%	20.80%

Table 2. Compression ratios using SCMPPM with $k = 5$ and $k = 7$, against plain PPM with $k = 7$. The ratios shown are average values over three collections.

Size	SCMPPM $k = 5$	MG System	XMill	XMLPPM	SCMHuff
1	24.89%	34.22%	36.46%	25.38%	39.34%
5	22.78%	30.72%	36.44%	25.70%	32.76%
10	22.10%	29.93%	36.49%	25.79%	31.13%
20	21.55%	29.30%	36.51%	25.80%	29.75%
40	21.08%	28.83%	36.55%	25.88%	28.76%
60	20.84%	28.67%	36.61%	25.91%	28.36%
100	20.60%	28.54%	36.56%	25.90%	27.91%

Size	SCMPPM $k = 5$	compress	zip	gzip	bzip2
1	24.89%	41.08%	35.27%	35.26%	28.95%
5	22.78%	40.80%	35.75%	35.66%	28.94%
10	22.10%	40.75%	35.81%	35.81%	26.37%
20	21.55%	40.70%	35.79%	35.79%	26.39%
40	21.08%	40.74%	36.01%	36.01%	26.54%
60	20.84%	40.58%	35.78%	35.89%	26.44%
100	20.60%	40.66%	35.91%	35.91%	26.43%

Table 3. Comparison between SCMPPM and other systems, using default settings for all. The ratios shown in the table are average values for each collection size, over the different collections tested.

of memory, and by at least 25% with respect to alternative state-of-art compressors.

The prototype is a basic implementation and we are working on several improvements, which will make it even more competitive. We are working to use words and separtors as base symbols of the PPM model, as words reflect much better than characters the true entropy of the text [TB90]. For example, a semiadaptive Huffman coder over the model that considers characters as symbols typically obtains a compressed file whose size is around 60% of the original size, on natural language. A Huffman coder when words are the symbols obtains 25% [ZMNBY00]. Another example is the WLZW algorithm (Ziv-Lempel on words) [BSTW86, DPS99].

We also plan to investigte more in depth the relationship between the type and density of the structuring and the improvements obtained with our method, since ours success is based on a semantic assumption and it would be interesting to see how this works on other text collections. We also plan to experiment with XML collections (INEX, DBLP, IMDB, SwissProt and others) to verify the benefits of the technique.

We note in particular that SCMPPM is equivalent to having an extra context given by the last structure tag seen. This extra context comes before the first character context. Under this light, we envision serveral generalizations, such as using more than one structural context (e.g. the two last structural elements containing the current text), and interleaving the structural with the character contexts in other orders than first the structural and then the character contexts.

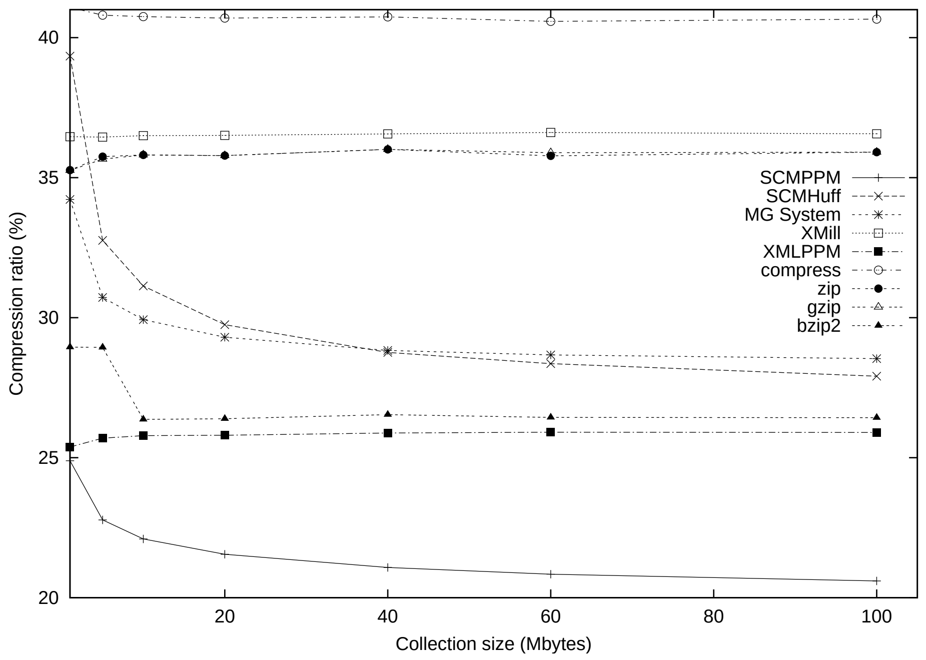

Figure 3. Graphical comparison between SCMPPM and other systems

References

[ANdlF03] J. Adiego, G. Navarro, and P. de la Fuente. SCM: Structural contexts model for improving compression in semistructured text databases. In *Proc. String Processing and Information Retrieval (SPIRE'03)*, LNCS 2857, pages 153–167. Springer, 2003.

[BSTW86] J. Bentley, D. Sleator, R. Tarjan, and V. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29:320–330, 1986.

[Che01] J. Cheney. Compressing XML with multiplexed hierarchical PPM models. In *Proc. Data Compression Conference (DCC 2001)*, pages 163–, 2001.

[CW84] J. Clearly and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, COM-32(4):396–402, April 1984.

[DPS99] J. Dvorský, J. Pokorný, and V. Snásel. Word-based compression methods and indexing for text retrieval systems. In *ADBIS'99*, LNCS 1691, pages 75–84. Springer, 1999.

[Har95] D. Harman. Overview of the Third Text REtrieval Conference. In *Proc. Third Text REtrieval Conference (TREC-3)*, pages 1–19, 1995. NIST Special Publication 500-207.

[Huf52] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proc. Inst. Radio Engineers*, 40(9):1098–1101, 1952.

[LS00] H. Liefke and D. Suciu. XMill: an efficient compressor for XML data. In *Proc. ACM SIGMOD 2000*, pages 153–164, 2000.

[TB90] I. Witten T. Bell, J. Cleary. *Text Compression*. Prentice Hall, Englewood Cliffs, N.J., 1990.

[WMB99] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, Inc., second edition, 1999.

[ZL77] J. Ziv and A. Lempel. An universal algorithm for sequential data compression. *IEEE Trans. on Information Theory*, 23(3):337–343, 1977.

[ZMNBY00] N. Ziviani, E. Moura, G. Navarro, and R. Baeza-Yates. Compression: A key for next-generation text retrieval systems. *IEEE Computer*, 33(11):37–44, November 2000.