# Tradeoffs in XML Database Compression

James Cheney
University of Edinburgh
Edinburgh EH8 9LE, United Kingdom
+44(0)131 651 3842
`jcheney@inf.ed.ac.uk`

**Abstract**

Large XML data files, or *XML databases*, are now a common way to distribute scientific and bibliographic data, and storing such data efficiently is an important concern. A number of approaches to XML compression have been proposed in the last five years. The most competitive approaches employ one or more statistical text compressors based on PPM or arithmetic coding in which some of the context is provided by the XML document structure. The purpose of this paper is to investigate the relationship between the extant proposals in more detail. We review the two main statistical modeling approaches proposed so far, and evaluate their performance on two representative XML databases. Our main finding is that while a recently-proposed multiple-model approach can provide better overall compression for large databases, it uses much more memory and converges more slowly than a single-model approach.

## 1 Introduction

Over the last few years, XML has become popular as an exchange format for large data collections, including scientific databases like the Georgetown Protein Structure Database (`pir.georgetown.edu`) and UniProt (`www.ebi.uniprot.org`) as well as bibliographic databases such as Medline (`www.ncbi.nlm.nih.gov`) and DBLP (`dblp.uni-trier.de`). We refer to such documents as *XML databases*. Due to the high redundancy of XML's text representation, compression is clearly needed for storing and transmitting XML databases efficiently. Although there is no obstacle to using general-purpose text compressors to compress XML (and this is what is done most frequently in practice), several researchers have proposed XML-specific compression techniques which take advantage of the structure of XML to improve compression beyond what is typically achieved by text compressors alone.

The first such effort was Liefke and Suciu's XMill [8], a compressor which splits the text of the XML document into *containers* and compresses each container using a text compressor, such as `gzip`, `bzip2` or PPM [5]. XMill is fast and typically provides compression improvement of around 20% compared to `gzip`, but typically does not compress better than bzip2 or PPM. In addition, XMill runs "offline", that is, it processes the document in large chunks, rather than processing it one symbol at a time. Cheney [3] developed xmlppm, a streaming compressor that uses a modeling technique called *multiplexed hierarchical modeling* (MHM). MHM switches among a small number of PPM models, one for element, attribute, character, and miscellaneous data, and "injects" element context symbols into the other models to recover accuracy lost due to model splitting. xmlppm typically provides compression performance improvements of 10–25% over the best achieved by XMill or bzip2. Adiego, de la Fuente, and Navarro [1] developed SCMPPM, a variant of the xmlppm approach which uses a technique called *structural context modeling* (SCM): rather than switching among a small number of models based on the syntactic class of the data, SCMPPM uses a separate model to compress the content under each element symbol. Leighton, Diamond and Müldner [7] developed AXECHOP, which uses XMill's container approach for text content and grammar-based compression to encode the element structure of the document. Hariharan and Shankar [6] presented XAUST, a compressor which takes advantage of DTD information to compress the element structure and which uses the structural context model to compress character data (albeit using order-4 arithmetic coding rather than PPM). Cheney [4] developed dtdppm, a DTD-conscious extension of xmlppm, and reported experiments indicating that DTD information provides compression improvements in the neighborhood of 20–40% relative to xmlppm for small, highly structured documents, but little improvement for large documents.

Although several approaches have been proposed, we believe that not enough careful empirical evaluation of the proposals has been performed. For example, both SCMPPM and XAUST were compared with xmlppm and compression improvements in the neighborhood of 20% were reported. While it is certainly plausible that the structural context model approach underlying both techniques provides significant compression benefits relative to xmlppm, we believe that the experimental results in the above papers do not provide enough evidence to establish this conclusively.

One reason for skepticism is that the version of xmlppm used in their experiments uses a single 1MB model for character data, while SCMPPM uses one 1MB model *per element symbol* and XAUST has no memory limit. For large documents, both SCMPPM and XAUST typically use considerably more than 1MB of model memory, so it is unclear whether the reported compression improvements are due to more accurate modeling or the use of more memory. In addition, the SCMPPM approach may allocate much more memory than it actually uses, which may limit the amount of memory it can use effectively.

A second objection to the experimental methodology used to evaluate SCMPPM and XAUST is that both only considered overall compression performance for very large documents (from 1MB to 1GB range). Because it splits text among many models, the SCM modeling technique seems likely to converge to its minimum bit rate more slowly than either MHM or plain PPM. Although rapid convergence is not important for compressing large XML databases, it may be important for compressing large collections of small XML documents; this may be a more realistic scenario for employing compression in an information-retrieval setting.

Of course, to some extent both criticisms apply to the experiments in Cheney [3, 4] as well, since memory use was not taken into account there either and the earlier paper did not evaluate the performance of `xmlppm` for large documents (in part because few XML databases were publicly available at the time). In this paper, we will remedy these omissions.

The purpose of this paper is to compare the proposed statistical modeling techniques carefully enough to be able to draw some conclusions about their relative performance in different situations. To simplify matters, we argue that the dominant factor in XML database compression is the method used for compressing unstructured text. We present the results of experiments comparing plain PPM, MHM, and SCM on two representative XML databases. Our experiments measure the memory utilization (amount of allocated memory that is actually used), the tradeoff of memory use vs. compression rate, and the convergence rate (the relationship between amount of input data seen and compression rate). We also present and evaluate a hybrid approach that attempts to combine the good features of MHM and SCM.

Our main findings are as follows. Neither approach is better in all situations. The SCM approach does, as previous experiments suggest, provide compression benefits over MHM for large files. However, these benefits come at a high memory cost. SCM has very low memory utilization, and performs significantly worse than either MHM or PPM when less than 10MB of model memory is available. In addition, SCM converges to its limiting compression rate more slowly than MHM, so is less suitable for compressing small or nonuniform files. The hybrid approach we propose avoids the worst-case behavior of the MHM and SCM models, but we believe that further improvements are possible.

The rest of this paper is structured as follows. We first review the MHM and SCM modeling techniques in detail, and present a hybrid technique. We then present the experimental methodology and analyze experimental results concerning the memory use and convergence behavior of the approaches. We conclude with a discussion of future directions.

## 2 XML compression models

In this paper, we focus on statistical approaches to XML compression, as employed by `xmlppm`, `dtdppm`, SCMPPM, and XAUST, since these techniques have the best published results. We first make and justify two simplifying assumptions: we will focus only on compression performance for the text content of XML documents, and we will ignore the issue of whether a DTD is used. These assumptions dramatically decrease the number of combinations of features of the above four compressors we need to compare, and reduce the problem to comparing the effectiveness of the SCM and MHM models when encoding the text content of an XML document.

We believe that the technique used for compressing unstructured text in the dominant factor affecting XML database compression performance. As Table 1 shows, `gzip` reduces the structure of typical XML databases to 10–19% of the size of the whole compressed document, whereas an XML-conscious compressor such as `xmlppm` typically compresses the element content to 1–4% of the total compressed size. Signif-

| | gzip | | | xmlppm | | |
|---|---|---|---|---|---|---|
| file | struct | total | %struct | struct | total | %struct |
| DBLP | 9.9MB | 52.4MB | 19% | 667KB | 33.4MB | 2.0% |
| Medline | 2.7MB | 20.2MB | 14% | 539KB | 13.7MB | 3.9% |
| XMark | 4.1MB | 38.1MB | 11% | 287KB | 27.6MB | 1.0% |
| PSD | 13.6MB | 108MB | 12% | 2.5MB | 79.6MB | 3.1% |

Table 1: Structure cost for `gzip` and `xmlppm` for several XML databases. For each compressor, the first two columns report the compressed size of the structural data and total compressed size. The last column shows the cost of structure as a percentage of the compressed document size.

icant improvments to XML database compression can only take place by improving the encoding of text content. Therefore, we will focus exclusively on how text is encoded by the various approaches.

Two compressors, `dtdppm` and XAUST, make use of DTD information. DTDs primarily provide information about the element and attribute structure, and provide little information about text content. There is one important exception: in the presence of a DTD, formatting whitespace (indentation) can be detected and ignored. Doing so can be very beneficial to PPM-based compressors because indentation tends to flush the PPM context (as observed by Cheney in evaluating `dtdppm` [4]). However, such whitespace is usually already removed in large XML databases. DTDs do not provide any other opportunities for improving text content compression. So for our purposes, it is irrelevant whether the compressor has access to a DTD.

`xmlppm` and `dtdppm` use essentially the same technique for compressing text, namely MHM. Similarly, both SCMPPM and XAUST use essentially the same approach, namely structural context modeling. We now describe the exact behavior of the four approaches we shall evaluate.

**Plain PPM** An obvious approach to compressing the text in an XML document is simply to compress it using a single PPM model. This corresponds to the *multiplexed modeling* technique used in an early version of `xmlppm` [3]. PPM shall serve as our performance baseline.

**Multiplexed Hierarchical Modeling** The MHM approach switches among several PPM models, one for element, attribute, character, and miscellaneous content. So, it uses a single PPM model to compress all of the character data in a document. In addition, in MHM, whenever the element context changes, a bytecode symbol representing the new element context is *injected* into the text model. This means that the bytecode is reported to the PPM model, so that the context and statistics are updated, but the symbol is not encoded. It is only safe to inject symbols that the decoder will also be able to inject at the corresponding point during decoding. In `xmlppm`, the element structure is encoded in a separate PPM model, so the surrounding element bytecode is always available to the decoder when it is needed.

As an example, consider the XML document shown in Figure 1. Assuming that the elements `book`, `title`, `author` are encoded as bytecodes `00`, `01`, and `02` respectively, the encoding of the character data of this document in MHM is

```
<book>
  <title>Gone with the Wind</title> <author>Margaret Mitchell</author>
  <chapter>...</chapter> ... <chapter> ... </chapter>
</book>
```

Figure 1: Simple XML example

```
(00) "\n  " (01) "Gone..." (00) " " (02) "Marg..." (00) "\n"
```

Here, string literals `"abc"` abbreviate null-terminated sequences `'a' 'b' 'c' \0`, and injected bytecodes are enclosed in parentheses. When a start tag is encountered, its code is injected, while when an end tag is encountered, its parent's code is injected. Also, whitespace must be preserved, since in the absence of a DTD, we have no way to know whether whitespace is significant.

The motivation for symbol injection is to give the PPM compressor "hints" concerning the surrounding element context. These hints can aid compression by priming the compressor to deal with whatever text is coming next. For example, the bytecode for `book` is highly correlated with whitespace. On the other hand, because the injected bytecodes are treated just like ordinary characters by the PPM model, this approach may lose accuracy because symbols have multiple meanings.

**Structural Context Modeling** In structural context modeling, one PPM model $M(e)$ is allocated for each element symbol $e$. (In a real compressor, an additional model is required to compress any text outside the main XML document tree, such as comments; we ignore this issue). As in MHM, the compressor switches between models on the basis of the XML content. However, instead of using a single model for all text, SCM uses a different model $M(e)$ to compress all text immediately enclosed in $e$. As an example, the XML document of Figure 1 is encoded as follows:

```
M(book)      "\n  "           " "            "\n"
M(title)           "Gone..."
M(author)                         "Marg..."
M(chapter)                                       "..."
```

Here, each line corresponds to a model, and text is encoded from left to right. The motivation for model splitting in SCM is the same as for symbol injection in MHM: text enclosed in the same element context is likely to be similar, so using separate models should provide compression benefits. Nevertheless, since each model must adapt to the distribution of its text independently, this approach also has the potential drawback that separate models that contain statistically similar text may duplicate effort.

**Hybrid Context Modeling** As we shall observe later, MHM tends to converge rapidly and require little memory, while SCM converges slowly, needs more memory, but compresses significantly better than MHM can. Neither model dominates in terms of compression vs. memory use or convergence rate. This motivates consideration of modeling techniques that combine the advantages of MHM and SCM.

We propose a hybrid approach called Hybrid Context Modeling (HCM). In this approach, we initially use a single model with context symbol injection, as in MHM.

In addition, we track the number of symbols seen in each element context. When this number exceeds a given bound $m$, the element context is considered "mature", and is allocated its own PPM model, as in SCM. In any sufficiently large file, many elements are likely to mature, so to limit memory use, we place a limit $n$ on the number of separate models. Only the first $n$ contexts to mature are allocated models.

We evaluated several combinations of $m$ and $n$ parameters, and found that HCM with $m = 10000, n = 8$ tends to perform well; we used these parameters in the reported experiments.

## 3 Experimental methodology

We evaluated the compression performance of PPM, MHM, SCM, and HCM models for variety of memory sizes and a variety of input sizes. We implemented each modeling technique on top of the PPMII implementation of PPM by Dmitry Shkarin [11]. PPMII provides very effective text compression and very good performance, and is not difficult to modify to support symbol injection. Each implementation encodes only the text data of the XML document; additional element structure that would be needed to decompress the document is not encoded. Thus, the results we report are estimates which we believe predict overall compression behavior.

Since all of the models rely on PPM compression, the running time differences between the models did not appear to be significant, except perhaps for incidental reasons such as cache or virtual memory behavior. In particular, the observed overhead of symbol injection in MHM is small, since the number of injected element symbols is generally much smaller than the number of text symbols. We used models with maximum order 5 in all the experiments; informal experiments with larger or smaller orders exhibited similar trends.

We evaluated the modeling techniques in three ways. First (Figure 2), we measured the memory utilization of the compressors, that is, the amount of memory actually used vs. the amount allocated. Second (Figure 3), we measured compression rate vs. amount of memory used. Finally (Figure 4), we measured the convergence behavior of the compressors, that is, the relationship between the compressed bit rate and the amount of input data seen.

Measuring the amount of memory actually used by these models requires care. The reason is that the PPMII models each allocate a block of memory and manage it manually, rather than using `malloc` and `free`; when memory runs out, the model is restarted. Not all of this memory is necessarily used by the model; in particular, in SCM or HCM elements with little text content will never use up their allocated memory, and modern operating systems typically do not allocate physical pages for virtual memory that is not used. Thus, it is not fair to estimate the memory used by a model by its total allocated memory size. Instead, we measured the *resident set size* (as reported by the operating system) at the end of compression. This includes all memory used by the program, not just by the PPM models. To obtain a variety of memory usage scenarios, we considered PPM model sizes from 4KB to 256MB in increasing powers of two; SCM ran out of memory for models larger than 32MB.

The convergence rate experiments measured compression (bits per input character) for each model using the first $10, 20, 50, 100, 200, 500, \ldots$ lines of the source files. For each model, we used the largest possible memory setting, in order to delay convergence as long as possible and minimize the effect of model restarts.

We used two large database files as benchmarks. The first, `dblp.xml` (300MB), is an XML version of the DBLP bibliographic database. DBLP has flat, regular structure, and is representative of flat or relational XML datasets. The second, `psd7003.xml` (717MB), is part of the Georgetown Protein Sequence Database (PSD). PSD consists of irregularly-structured records describing properties of proteins along with metadata such as journal citations. PSD has deeper and more irregular structure than DBLP, and is representative of XML databases used in bioinformatics.

We experimented with additional real XML databases, including data from Medline (a medical bibliographical database) and UniProt (a bioinformatics database), as well as synthetic XML database benchmarks [2, 9, 10]. However, the real databases behaved similarly to either DBLP or PSD, while none of the synthetic databases behaved like any real databases. We believe this is because the textual context of the synthetic databases is randomly generated; indeed, all the synthetic benchmarks are distributed as relatively small C/C++ programs that generate large XML files, so compression is not a pressing issue for such databases. As a result, we do not believe the synthetic benchmarks make useful compression benchmarks.

All experiments were performed on an AMD Athlon 3000+ (clock speed 1.8 Ghz, 512MB RAM) running Fedora Core 3.

## 4 Analysis

**Memory utilization** Figure 2 plots the memory utilization of the four methods, along with the curve for 100% utilization. Memory utilization for PPM and MHM is near 100% until the model size exceeds the memory needs of the document. The SCM approach allocates one model of size $M$ for each of $k$ observed element symbols, for a total memory allocation of $kM$ bytes. In practice this upper limit is rarely reached: most of the text is usually located in one of a small number of contexts. The HCM approach (with $n = 8$) allocates $9M$ bytes; as with SCM, this limit is rarely reached, but memory utilization is significantly higher than for SCM.

**Memory vs. bit rate** Figure 3 plots the compressed bit rate (bits per character) vs. amount of memory used. First, we note that MHM always performs better (in the limit, 5–10% better) than PPM using the same amount of memory. On the other hand, SCM performs worse than PPM until 3–10MB, and performs worse than MHM up to 10–20MB. Nevertheless, eventually SCM performs up to 5–10% better than MHM ever does. Although HCM performs slightly better than either MHM or SCM at one or two points, it compresses worse than MHM for low memory and worse than SCM for high memory.

**Convergence rate** Figure 4 plots the bit rate vs. amount of input data seen. Interestingly, PPM has the best performance for short documents, followed by MHM; we believe this is because the element bytecodes injected by MHM initially interfere

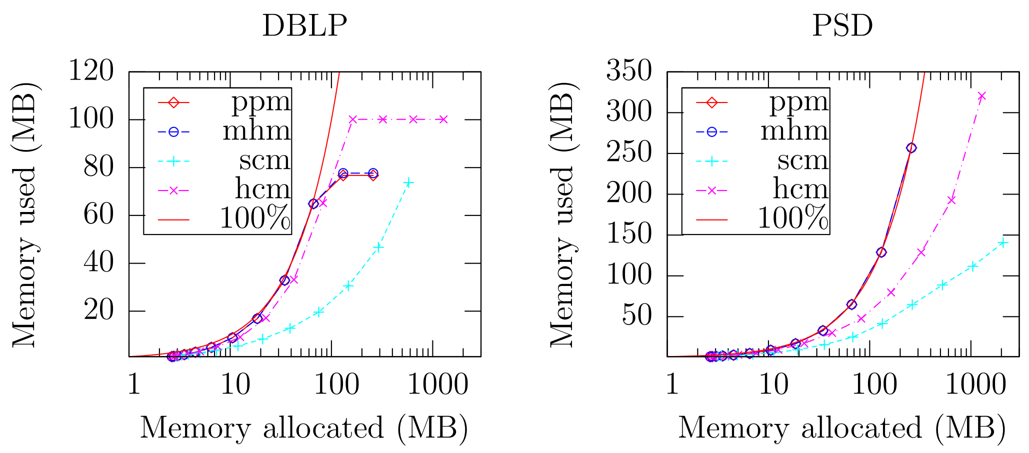

Figure 2: Memory utilization

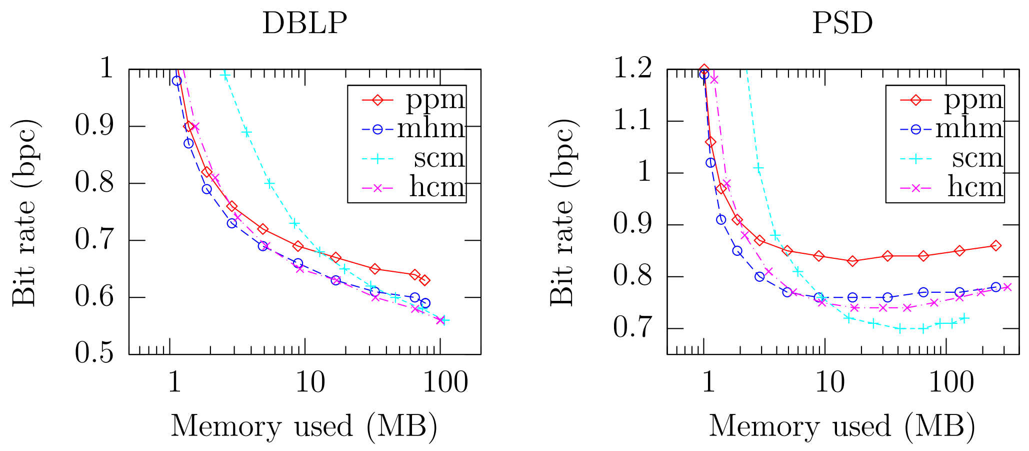

Figure 3: Memory use vs. compression

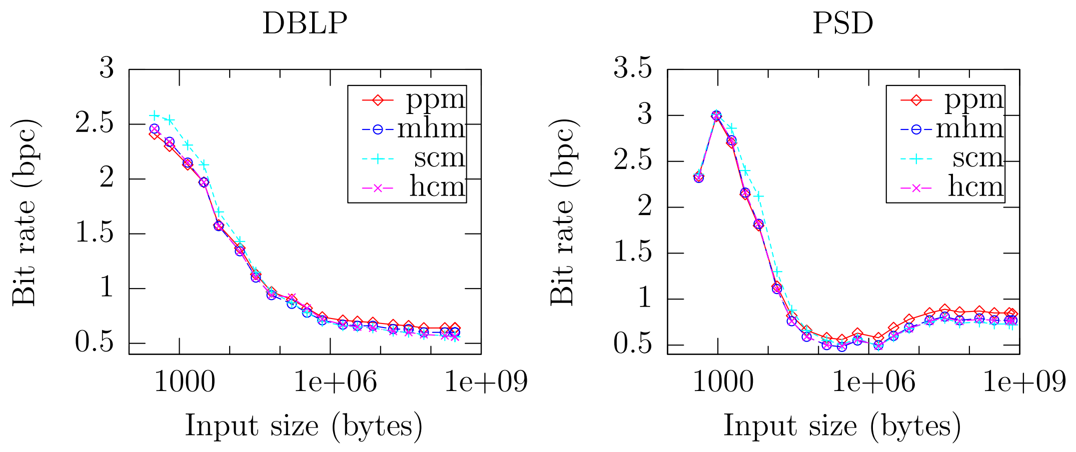

Figure 4: Convergence rate

with compression. The SCM model performs up to 5% worse initially, but catches up at around the 100KB–1MB range. Again, HCM behaves like MHM initially and then behaves like SCM. For DBLP, HCM exhibits a slight "bump" at around the 10–50KB mark, presumably because of a brief decrease in coding effectiveness as element contexts mature and are allocated separate models. This can probably be avoided by training the new models before they are used.

## 5 Future work

Although DTDs provide no information useful for improving text content compression in typical XML databases, other schema systems such as XML Schema or RELAX/NG can provide considerably more detail about text content, such as regular expression or semantic constraints. This information should be useful in improving XML database compression.

To keep the number of experiments and results manageable, we varied the amount of input data seen and amount of memory used, but kept the model order fixed. In informal experiments, varying the model order did not appear to affect the overall trends we observed. The relationship between model order, memory use, and compression performance needs to be further investigated in order to find the best tradeoffs for a practical compressor.

Our experiments show that neither MHM nor SCM is better in all situations. HCM converges more rapidly and uses less memory than SCM while providing better compression than MHM when enough memory is available. However, HCM seldom performed better than both MHM and SCM, and we believe there is still a lot of room for improvement. Essentially, the problem is that neither MHM nor SCM is sufficiently adaptive to the XML context. We believe that a logical next step is to develop a model that can take better advantage of XML context information, without giving up on the benefits of sharing among contexts and without wasting memory. One possibility is to dynamically cluster or resize models based on additional statistics. Another possibility might be to modify the underlying PPM models to take XML context into account. However, this is not a step to be taken lightly, since there are many benefits to using existing (and well-optimized) implementations such as PPMII "off the shelf".

## 6 Conclusions

XML has become a popular format for online distribution of large datasets. Because XML markup is highly redundant, compression is necessary for efficient storage and transmission of such data. Several XML-specific compression techniques have been proposed, but little careful experimentation comparing the proposals has been performed. In this paper, we have compared four approaches, each of which uses PPM models to compress the text in XML documents, and three of which also leverage the element context of XML to improve compression. We have confirmed claims by other researchers that the SCM approach provides the best overall compression for large documents, but observed that it requires much more memory to obtain this performance, and performs poorly on short documents. In contrast, the MHM model proposed in our earlier work has reasonable performance for low memory situations and small documents, yet performs within 5–10% of the best compression achieved by SCM. We also proposed and evaluated a hybrid approach called HCM, but found that it never performs significantly better than both MHM and SCM. These experiments should guide research towards improvements in XML compression.

## References

[1] J. Adiego, P. de la Fuente, and G. Navarro. Merging prediction by partial matching with structural contexts model. In *Proceedings of the 2004 IEEE Data Compression Conference (DCC 2004)*, page 522, 2004.

[2] Stéphane Bressan, Mong Li Lee, Ying Guang Li, Zoé Lacroix, and Ullas B. Nambiar. *XML Data Management*, chapter 17: XML Management System Benchmarks. Addison-Wesley, 2003.

[3] James Cheney. Compressing XML with multiplexed hierarchical models. In *Proceedings of the 2001 IEEE Data Compression Conference (DCC 2001)*, pages 163–172. IEEE, 2001.

[4] James Cheney. An empirical evaluation of simple DTD-conscious compression techniques. In *Proceedings of the Eighth Workshop on the Web and Databases (WebDB 2005)*, pages 43–48, 2005.

[5] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Comm.*, COM-32(4):396–402, 1984.

[6] S. Hariharan and Priti Shankar. Compressing XML documents with finite state automata. In *Proceedings of the Tenth International Conference on Implementation and Application of Automata (CIAA 2005)*, 2005. Formal proceedings to appear.

[7] Gregory Leighton, Jim Diamond, and Tomasz Müldner. AXECHOP: A grammar-based compressor for XML. In *Proceedings of the 2005 IEEE Data Compression Conference (DCC 2005)*, page 467, 2005.

[8] Hartmut Liefke and Dan Suciu. XMill: An efficient compressor for XML data. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on management of data*, pages 153–164. ACM Press, 2000.

[9] Jignesh M. Patel and H. V. Jagadish. *XML Data Management*, chapter 18: The Michigan Benchmark. Addison-Wesley, 2003.

[10] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML benchmark project. Technical Report INS-R0103, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, 2001.

[11] Dmitry Shkarin. PPM: One step to practicality. In *Proceedings 12th IEEE Data Compression Conference (DCC 2002)*, pages 202–211, 2002.