XML compression techniques: A survey and comparison

Sherif Sakr 1

National ICT Australia (NICTA), 223 Anzac Parade, NSW 2052, Sydney, Australia

1. Introduction

The eXtensible Markup Language (XML) has been acknowledged to be one of the most useful and important technologies that has emerged as a result of the immense popularity of HTML and the World Wide Web. Due to the simplicity of its basic concepts and underlying theories, XML has been used in solving numerous problems such as providing neutral data representation between completely different architectures, bridging the gap between software systems with minimal effort and storing large volumes of semi-structured data. XML is often referred to as self-describing data because it is designed in a way that the schema is repeated for each record in the document. On one hand, this self-describing feature provides XML with immense flexibility but on the other hand, it also introduces the main problem of verbosity of XML documents which results in huge document sizes. This huge size lead to the fact that the amount of information that has to be transmitted, processed, stored, and queried is often larger than that of other data formats. Since XML usage is continuing to grow and large repositories of XML documents are currently pervasive, a great demand for efficient XML compression tools has been exist. To tackle this problem, several research efforts have proposed the use of XML-conscious compressors which exploits the well-known structure of XML documents to achieve compression ratios that are better than those of general text compressors. The usage of XML compression tools has many advantages such as: reducing the network bandwidth required for data exchange, reducing the disk space required for storage and minimizing the main memory requirements of processing and querying XML documents.
Experimental evaluation and comparison of different techniques and algorithms which deals with the same problem is a crucial aspect especially within the applied domains of computer science. In this paper, we provide a complete survey of the current state-of-the-art of XML compression techniques and an extensive experimental study of the available implementations of these techniques. Specifically, we make the following contributions:

- We present a comprehensive survey of the state-of-the-art of XML compression techniques.
- We collect and construct a rich XML corpus which contains a wide variety of XML data sources, natures and document sizes. This corpus establishes a good basis for assessing the characteristics of XML compression tools.
- We present the detailed results of examining the performance characteristics of nine publicly available XML compression tools.
- For ensuring repeatability as one of the main targets of this work, the web page of this study [1] provides access to the test files, examined XML compressors and the detailed results of this study.

The remainder of this paper is organized as follows. Section 2 briefly introduces the different classifications of XML compression tools. Sections 3, 4 and 5 present our survey over the different classes of XML compression techniques. Section 6 presents the characteristics and data sets of our XML corpus used to perform our experiments. Detailed and consolidated results of our experiments are presented in Section 7. Section 8 presents our suggested ranking of the examined compressors before we draw our final conclusions in Section 9.

2. Features and classifications of XML compression tools

A very large number of XML compressors have been proposed in the literature of recent years. These XML compressors can be classified with respect to two main characteristics. The first classification is based on their awareness of the structure of the XML documents. According to this classification, compressors are divided into two main groups:

- **General Text Compressors**: Since XML data are stored as text files, the first logical approach for compressing XML documents was to use the traditional general purpose text compression tools. This group of XML compressors [6,2,26] is XML-Blind, i.e. they treat XML documents as usual plain text documents and thus apply the traditional text compression techniques [43].

- **XML Conscious Compressors**: This group of compressors are designed to take the advantage of the awareness of the XML document structure in order to achieve better compression ratios over the general text compressors. This group of compressors can be further classified according to their dependence on the availability of the schema information of the XML documents as follows:
  - **Schema dependent compressors** where both of the encoder and decoder must have access to the document schema information to achieve the compression process [8,12,29].
  - **Schema independent compressors** where the availability of the schema information is not required to achieve the encoding and decoding processes [38,9,19,24].

Although schema dependent compressors may be, theoretically, able to achieve slightly higher compression ratios, they are not preferable or commonly used in practice because there is no guarantee that the schema information of the XML documents is always available.

The second classification of XML compressors is based on their ability of supporting queries.

- **Non-Queriable (Archival) XML Compressors**: This group of the XML compressors does not allow any queries to be processed over the compressed format [38,5,9,12,24]. The main focus of this group is to achieve the highest compression ratio. By default, general purpose text compressors belong to the non-queriable group of compressors.

- **Queriable XML Compressors**: This group of the XML compressors allow queries to be processed over their compressed formats [14,39,44]. The compression ratio of this group is usually worse than that of the archival XML compressors. However, the main focus of this group is to avoid full document decompression during query execution. In fact, the ability to perform direct queries on compressed XML formats is important for many applications which are hosted on resource-limited computing devices such as mobile devices and GPS systems. By default, all queriable compressors are XML conscious compressors as well. This group of compressor can be further classified according to their way of encoding the structural and data parts of the XML documents as follows:
  - **Homomorphic compressors** where the original structure of the XML document is retained and the compressed format can be accessed and parsed in the same way of the original format [14,40].
  - **Non-homomorphic compressors** where the encoding process of the XML document serrates the structural part from the data part. Therefore, the structure of the compressed format is different from the structure of the original XML document [22,36,39,44].
3. General text compressors

XML is a text representation for a tree structured data. Hence, a straightforward logical approach for compressing XML documents is to use the traditional general purpose text compression tools. Numerous algorithms have been devised over the past decades to efficiently compress text data. The most popular and efficient representatives of this group are: gzip [6], bzip2 [2] and PPM [26] compressors.

The gzip compressor is based on the DEFLATE lossless data compression algorithm which uses a combination of the LZ77 algorithm [52] and Huffman coding [31]. LZ77 algorithm achieves compression by replacing portions of the data with references to matching data that has already passed through both encoder and decoder. Huffman coding uses a specific method for choosing the representation for each symbol where the most common characters using shorter strings of bits than are used for less common source symbols.

The bzip2 compressor uses the Burrows-Wheeler transform [23] to convert frequently recurring character sequences into strings of identical letters, and then applies a move-to-front transform and finally Huffman coding. The Burrows-Wheeler transform permutes the order of the characters in a way that if the original string had several substrings that occurred often, then the transformed string will have several places where a single character is repeated multiple times in a row. This is useful for compression, since it tends to be easy to compress a string that has runs of repeated characters. The bzip2 compression compresses files at a higher compression ratio than those compressed using gzip but it also has slower performance.

PPM is an adaptive statistical data compression technique based on context modeling and prediction. It uses a finite-context statistical modeling technique that can be viewed as blending together several fixed-order context models to predict the next character in the input sequence. Prediction probabilities for each context in the model are calculated from frequency counts which are updated adaptively. The symbol that actually occurs is encoded relative to its predicted distribution using arithmetic coding. Although PPM is simple and is the most efficient of the compressors presented so far, it is also computationally the most expensive.

4. Non-queriable (archival) XML compressors

4.1. Schema independent compression schemes

In [38] Liefke and Suciu have presented the first implementation of an XML conscious compressor. XMill has introduced some novel ideas for XML conscious compression which are followed by other XML compressors. The most important ideas are that of separating the structure from data and the grouping of the data values into homogenous containers based on their relative paths in the tree and their data types. In XMill, both of the structural and data value parts of the source XML document are collected and compressed separately. In the structure part, XML tags and attributes are encoded in a dictionary-based fashion before passing it to a back-end general text compression scheme. The structural encoding scheme of XMill assigns each distinct element and attribute name an integer code, which serves as the key into the element and attribute name dictionaries. In the data part, data values are grouped into homogenous and semantically related containers according to their path and data type. Each container is then compressed separately using specialized compressor that is ideal for the data type of this container. This grouping operation serves to localize repetitions and therefore enhance the degree of compression. In the latest versions of the XMill source distribution, the intermediate binaries of the compressed format can be passed to one of three alternative back-end general purpose compressor: gzip, bzip2 and PPM. In [45] Skibinski and Swacha have presented the XWRT compressor which applies nearby similar ideas of XMill. It uses a dictionary-based compression technique called XML Word Replacing Transform. The idea of this technique is to replace the frequently appearing words with references to the dictionary which is obtained by a preliminary pass over the data. XWRT submits the encoded results of the preprocessing step to three alternative general purpose compression schemes: gzip, LZMA and PPM.

In [37], Li has described the implementation of another XML compressor which applies the similar idea of XMill with a slight change. In XComp, an XML document is parsed first and then its components are re-organized and forwarded to the compression engine. The information of the parsing phase is used to confine the maximum memory usage of containers. XComp claimed that the efficiency of memory usage can be increased by applying a memory window when the document container exceeds a set value.

In [24], Cheney has presented XMLPPM as a streaming XML compressor which uses a Multiplexed Hierarchical PPM Model called (MHM). XMLPPM is considered as an adaptation of the general purpose Prediction by Partial Matching compression scheme (PPM) [26]. In XMLPPM, an XML file is first parsed using an SAX parser to generate a stream of SAX events. Each event is encoded using a bytecode representation called ESAX (Encoded SAX). The ESAX bytecodes are encoded using one of several multiplexed PPM models based on its syntactic structure (elements, characters, attributes, and miscellaneous symbols). In [20] Adiego et al. described SCMPPM as a variant of the XMLPPM compressor. SCMPPM combines a technique called Structure Context Modelling (SCM) with the PPM compression scheme. It uses a bigger set of PPM models than XMLPPM as it uses a separate model to compress the text content under each element symbol.

In [48], Toman has presented the Exalt XML compressor which uses a syntactical-oriented approach for compressing XML documents. The Exalt compressor has inspired the idea of the Sequitur compression algorithm of the text data.
Table 1
Summary characteristics of archival XML compressors.

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Schema dependent</th>
<th>Compression Technique</th>
<th>Back-end compressor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMill [16]</td>
<td>NO</td>
<td>Dictionary-Based</td>
<td>Gzip, Bzip2, PPM</td>
</tr>
<tr>
<td>XWRT [19]</td>
<td>NO</td>
<td>Dictionary-Based</td>
<td>Gzip, LZMA, PPM</td>
</tr>
<tr>
<td>XComp [37]</td>
<td>NO</td>
<td>Dictionary-Based</td>
<td>Gzip</td>
</tr>
<tr>
<td>XMLPPM [17]</td>
<td>NO</td>
<td>Multiplexed Hierarchical PPM</td>
<td>PPM</td>
</tr>
<tr>
<td>SCMPM [9]</td>
<td>NO</td>
<td>Structure Context Modelling</td>
<td>PPM</td>
</tr>
<tr>
<td>AXECHOP [35]</td>
<td>NO</td>
<td>Context-Free Grammars</td>
<td>BWT + MPM</td>
</tr>
<tr>
<td>DTDPPM [3]</td>
<td>YES</td>
<td>Dictionary-Based</td>
<td>PPM</td>
</tr>
<tr>
<td>XAUST [12]</td>
<td>YES</td>
<td>Deterministic Finite Automata</td>
<td>Arithmetic Coding</td>
</tr>
<tr>
<td>rngzip [8]</td>
<td>YES</td>
<td>Deterministic Tree Automata</td>
<td>Gzip</td>
</tr>
<tr>
<td>Millau [29]</td>
<td>YES</td>
<td>Dictionary-Based</td>
<td>WBXML</td>
</tr>
</tbody>
</table>

and utilized the fact that XML document could be represented using a context-free grammar to dynamically infers a custom grammar for each document. The generated grammars use the grammar-based codes encoding technique introduced by Kieffer and Yang in [32] which is then encoded using adaptive arithmetic coding. In [35], Leighton et al. have presented another approach of grammar-based compressor for XML documents called AXECHOP. Following the idea of XMill, AXECHOP divides the source XML document into structural and data parts. However, for the structure part, AXECHOP uses a byte tokenization scheme that preserves the original structure of the document and then uses the MPM compression algorithm [33] to generate a context-free grammar which is represents this structure information. The generated grammar is then passed through an adaptive arithmetic coder before being written to the compressed file. In AXECHOP, the data part of the XML document is organized into a series of containers based on XML element or attribute names and then the Burrows–Wheeler Transform (BWT) [23] is applied to the contents of each container, with the results being appended to the compressed file.

4.2. Schema dependent compression schemes

In [29], Girardot and Sundaresan have presented Millau as the first XML schema dependent compressor. Millau does not require a Document Type Definition (DTD) for the processed XML document, but if present it can be used to build and optimize the token dictionaries in advance. The Millau encoding format is an extension of the WAP Binary XML format. The WBXML (Wireless Application Protocol Binary XML) Content Format Specification [10] defines a compact binary representation of XML. This format is designed to reduce the transmission size of XML documents with no loss of functionality or semantic information. Millau extends this format to adapt it to business applications while improving on the compression algorithm itself. It separates structure compression from text compression. Further, it takes advantage of the schema and data types to enable better compression.

In [34], League and Eng have presented a new tool for compressing XML documents that are conforming to a given Relax NG schema [7], RNGzip. In RNGzip, the sender and receiver must agree in advance on precisely the same schema. In this sense, the schema is like a shared key for encryption and decryption. RNGzip uses a Relax NG schema validator to build a deterministic tree automaton from the specified schema. Then given an XML document, it checks whether the XML is accepted by the automaton. Given this automaton, a receiver can reconstruct an entire XML document by transmitting very little information. If there is a choice point in the automaton, RNGzip just transmits which transition was taken and if it encounters the text transition then the matching text is transmitted. Table 1 summarizes the characteristics of the surveyed non-queriable (archival) XML compressors.

5. Queriable XML compressors

5.1. Homomorphic queriable XML compressors

In [47], Tolani and Haritsa have presented the first XML-conscious compression scheme to support querying without the need for a full decompression of the compressed XML document. XGrind does not separate data from structure. It retains the original structure of the XML document. The homomorphic nature of the XGrind compressed format grants XGrind many interesting features such as: 1) the compressed XML document can be viewed as the original XML document with its tags and element/attribute values replaced by their corresponding encodings. Hence, XGrind can be considered an extended SAX
parser. 2) XML indexing techniques can be built on the compressed document in a similar manner to those that can be built on regular XML documents. In XGrid, element and attribute names are encoded using a dictionary-based encoding scheme, and character data is compressed using semi-adaptive Huffman coding. The query processor of XGrid can only handle exact-match and prefix-match queries on compressed values and partial-match and range queries on decompressed values. However, several operations are not supported by XGrid, for example, non-equality selections in the compressed domain. Therefore, XGrid cannot perform any join, aggregation, nested queries, or construct operations.

In [40], Min et al. have described another homomorphic queriable XML compressor, XPress. It uses a combination of the characteristics to compress and retrieve XML data efficiently. It uses a reverse arithmetic encoding method over distinct character data value to support querying. The compression scheme of XPress is a semi-adaptive one which uses a preliminary scan of the input file to gather statistical information and the encoding rules for data are independent to the locations of data and it also uses proper encoders for data values based on their automatic inferred type information.

In [44], Skibinski and Swacha have presented the QXT compressor to extend their previous work in the XWRT compressor [45] with query-friendly concepts in order to make it possible to process queries with partial decompression. QXT works in two passes over the input XML document where the document is treated as an ordered sequence of \( n \) tokens. In the first pass, a dictionary is formed and the frequency of each of its items is computed. The complete dictionary is stored within the compressed file, so this pass is unnecessary during decompression. In the second pass, the actual transformation takes place; data are parsed into tokens, respectively encoded, and placed into separate containers, depending on their path from the document root. The containers are memory-buffered until they exceed a given threshold after which they are compressed with general-purpose compression algorithm and written to disk. The containers are compressed in 32 KB blocks, thus allowing partial decompression of blocks of such length. QXT uses a byte-oriented prefix code which is used to produce slightly longer output than, e.g., bit-oriented Huffman coding. However, the resulting data can be easily compressed further with the back-end general compressor. In QXT, query execution starts with reading the dictionary from the compressed file. Next, the query processor resolves which containers might contain data matching the query. The required containers are decompressed and the transformed data are then searched using the transformed pattern. Only the matching elements are decoded to the original XML form. QXT compressed format does not include any indices to keep the compression gain which had the top priority in the design of QXT.

5.2. Non-homomorphic queriable XML compressors

In [39], Lin et al. have proposed a grammar-based queriable XML compression scheme, XSeq. It is considered as an adaptation of the famous grammar-based text strings compression algorithm, Sequitur [41]. In XSeq, the tokens of the input XML file are separated into a set of containers each of which is then compressed using Sequitur. The Sequitur compression algorithm is a linear-time online algorithm that forms a context-free grammar for a given string input. Using the defined context-free grammars, XSeq avoids the sequential scan of irrelevant compressed data and only processes data values that are to be matched by the given query. In addition, context-free grammars allow XSeq to process queries directly over the compressed file without full or partial decompression. In order to correlate the data values stored in different containers and accelerate the query evaluation time, XSeq uses a structure of indices which are stored within the compressed file and are loaded into the memory before processing the rule contents. For example, it uses a structural index through which each data value can be quickly located in the container without decompression while the header index contains a list of pointers to the container buffer for each container in the file.

In [42], Ng et al. have presented the XCQ compressor. In this work, XCQ tried to exploit the information provided by a Document Type Definition (DTD) associated with an XML document to achieve better compression as well as generate more usable compressed data to support querying. XCQ is based on a DTD and SAX Event Stream Parsing technique called DSP. In this technique the compressed XML documents are stored in Partitioned Path-Based Grouping (PPG) data streams, which are equipped with a Block Statistics Signature (BSS) indexing scheme. The indexed PPG data streams support the processing of XML queries that involve selection and aggregation, without the need for full decompression. XCQ supports querying of compressed documents that conform to the Partitioned Path-Based Data Grouping by only partially decompressing them. The underlying idea used in the engine is that it only decompresses portions of the compressed document that are relevant to the query evaluation.

In [36], Leighton et al. have presented the TREECHOP XML compressor. The compression process in TREECHOP begins by conducting an SAX-based parsing of the XML document where the parsed tokens are written out to the compression stream in depth-first order. The codeword for each node is prefixed by its parent’s codeword and two nodes in the XML document tree share the same codeword if they have the same path. Each CDATA section, comment, processing instruction, and non-leaf node is assigned a binary codeword. This codeword is uniquely assigned based on the path of the tree node. Since tree node encodings are written to the compression stream in depth-first order, it is possible for the decompressor to regenerate the original XML document using the adaptive encoding information incrementally. In TREECHOP, exact-match and range queries can be carried out via a single scan through the compression stream.

The XQZip compressor [25] removes the duplicate structures in an XML document to improve query performance by using an indexing structure called the Structure Index Tree (or SIT). XQZip organizes the data values into a sequence of blocks before applying the gzip compression to each block which entails decompressing the entire blocks during query
evaluation. Thus, XQZip tries to determine the minimum number of blocks to be decompressed during the query evaluation. In addition, XQZip tries to reduce the decompression overhead in query evaluation by applying the Least Recently Used (LRU) cache algorithm to manage a buffer pool for the decompressed blocks of XML data. XQZip addresses a large portion of XPath queries such as multiple and deeply nested predicates with mixed value-based and structure-based query conditions, and aggregations; enriched with union and the grouping operator in the return step.

In [28] Ferragina et al. have presented the XBW transform that represents a succinct labeled tree using two arrays: the first contains the tree labels arranged in an appropriate order, while the second is a binary array encoding the structure of the tree. The XBW transform can be computed and inverted in linear time with respect to the number of tree nodes, and is as succinct as the information contained in the tree would allow. In [27], the authors adapted the XBW transform to derive a compressed searching and navigation tool for XML called the XZip compressor.

In [51], Wong et al. proposed a compact XML storage engine, called ISX. The storage layer of the ISX system consists of three layers: topology layer, internal node layer and leaf node layer. The topology layer uses a balanced parentheses encoding to store the tree structure of the XML document and facilitates fast navigational accesses. The internal node layer stores the XML elements, attributes and signatures of the text data for fast text queries. The leaf node layer stores the actual text data which can be compressed by any common compression techniques and referenced by the topology layer. The authors have presented a set of auxiliary (summary) data structures and algorithms over the ISX topology layer to support direct node navigation operators, to provide XPath query processing interface and to achieve efficient node insertion/deletion mechanism.

The XCpaqs compressor [49] starts by scanning XML document where structure and context are separated. During this scan, statistics of tag and path and recognition of path type are collected. In XCpaqs, the connection between structure and context is the order of path in the original document. For the structural part, tags are coded at first based on the collected statistics and then paths formed by tags are coded. The data coder encodes data with different path type with different code method where enumerate typed data is compressed by a dictionary compressor, string data is encoded with a suffix compressor, and long text is compressed by the BWT compression method [23]. The result of structure encoder and data coder are combined into a final structure of 2-ary as. Consequently, for each leaf in original XML document, there is a path code. And for a complex element, its structure and context is in related path codes. Due to this coding mechanism, the corresponding relationship between original XML document and compressed one can be easily recovered. The XCpaqs query processor translates tags in the query into their corresponding code based on tag code table, splits the query plan into three steps: selecting the appropriate path code from path table, relating the element and conditions according to their context, and finally constructing the final result. The XCpaqs query evaluation mechanism can handle queries in XQuery language which could be converted to the form query objects, conditions, construction. Query objects are the elements query to, conditions are the restrictions specified after the where sub-clause and construction is the clause to wrap query result to the given schema.

Finally, in [22], Arion et al. have presented the XQuec system. XQuec addresses the problem of compressing XML data in such a way as to allow efficient XQuery evaluation in the compressed format. It uses a fragmentation and storage model for the compressed XML documents which is based on the idea of separating structure and content within an XML document. In addition, it depends on a proper choice of how to group containers should ensure that containers belonging to the same group also appear together in query predicates. In order to perform the evaluation of a predicate within the compressed domain, it ensures that containers involved in the predicate belong to the same group and are compressed with an algorithm supporting that predicate in the compressed domain. The information about predicates is inferred using the available query workloads. XQueC exploits the query workload information to partition the containers into sets according to the source model and to properly assign the most suitable compression algorithm to each set [21,31]. XQueC has also designed an algebra for evaluating XML queries. This algebra is exploited by a cost-based optimizer which can freely mix regular operator and compression-aware ones. In addition, XQueC uses a variety of auxiliary structures, such as DataGuides [30], structure trees, and other indexes, in order to support efficient evaluation of XQuery. Table 2 summarizes the characteristics of the surveyed queriable XML compressors.

Table 2

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Homomorphic</th>
<th>Compression technique</th>
<th>Back-end compressor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGrind [14]</td>
<td>YES</td>
<td>Dictionary-Based</td>
<td>Huffman Encoding</td>
</tr>
<tr>
<td>XPress [40]</td>
<td>YES</td>
<td>Dictionary-Based</td>
<td>Reverse Arithmetic Encoding</td>
</tr>
<tr>
<td>QXT [44]</td>
<td>YES</td>
<td>Dictionary-Based</td>
<td>Gzip, LZMA, PPM</td>
</tr>
<tr>
<td>XSeq [39]</td>
<td>NO</td>
<td>Context-Free Grammars</td>
<td>Arithmetic Encoding</td>
</tr>
<tr>
<td>XCQ [42]</td>
<td>NO</td>
<td>Dictionary-Based</td>
<td>Block Statistics Signature</td>
</tr>
<tr>
<td>TREECHOP [36]</td>
<td>NO</td>
<td>Prefix-based Dictionary-Based</td>
<td>NO</td>
</tr>
<tr>
<td>XZip [25]</td>
<td>NO</td>
<td>Dictionary-Based</td>
<td>Gzip</td>
</tr>
<tr>
<td>XBZip [27]</td>
<td>NO</td>
<td>Succinct labeled tree</td>
<td>XBW</td>
</tr>
<tr>
<td>ISX [51]</td>
<td>NO</td>
<td>Succinct labeled tree</td>
<td>Gzip</td>
</tr>
<tr>
<td>XCPaq [49]</td>
<td>NO</td>
<td>Dictionary-Based</td>
<td>Dictionary-Based + Suffix Encoding + BWT</td>
</tr>
<tr>
<td>XQueC [18]</td>
<td>NO</td>
<td>Binary Path-Based Encoding</td>
<td>Divert based on types of data containers</td>
</tr>
</tbody>
</table>

Legend:
- Dictionary-Based
- Block Statistics Signature
- Arithmetic Encoding
- Huffman Encoding
- Reverse Arithmetic Encoding
- Gzip, LZMA, PPM
Table 3
Characteristics of XML data sets.

<table>
<thead>
<tr>
<th>Data set name</th>
<th>Document name</th>
<th>Size (MB)</th>
<th>Tags</th>
<th>Number of nodes</th>
<th>Depth</th>
<th>Data ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXI [4]</td>
<td>EXI-Telecomp.xml</td>
<td>0.65</td>
<td>39</td>
<td>651398</td>
<td>7</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>EXI-Weblog.xml</td>
<td>2.60</td>
<td>12</td>
<td>178419</td>
<td>3</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td>EXI-Invoice.xml</td>
<td>0.93</td>
<td>52</td>
<td>78377</td>
<td>7</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>EXI-Array.xml</td>
<td>22.18</td>
<td>47</td>
<td>1168115</td>
<td>10</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>EXI-Factbook.xml</td>
<td>4.12</td>
<td>199</td>
<td>104117</td>
<td>5</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>EXI-Geographic Coordinates.xml</td>
<td>16.20</td>
<td>17</td>
<td>55</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>XMark [15]</td>
<td>XMark1.xml</td>
<td>11.40</td>
<td>74</td>
<td>520546</td>
<td>12</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>XMark2.xml</td>
<td>113.80</td>
<td>74</td>
<td>5167121</td>
<td>12</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>XMark3.xml</td>
<td>571.75</td>
<td>74</td>
<td>25900899</td>
<td>12</td>
<td>0.74</td>
</tr>
<tr>
<td>XBench [13]</td>
<td>DCSD-Small.xml</td>
<td>10.60</td>
<td>50</td>
<td>6190628</td>
<td>8</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>DCSD-Normal.xml</td>
<td>105.60</td>
<td>50</td>
<td>6190628</td>
<td>8</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>TCSD-Small.xml</td>
<td>10.95</td>
<td>24</td>
<td>831393</td>
<td>8</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>TCSD-Normal.xml</td>
<td>106.25</td>
<td>24</td>
<td>8085816</td>
<td>8</td>
<td>0.78</td>
</tr>
<tr>
<td>Wikipedia [11]</td>
<td>EnWikiNews.xml</td>
<td>71.09</td>
<td>20</td>
<td>2013778</td>
<td>5</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>EnWikiQuote.xml</td>
<td>127.25</td>
<td>20</td>
<td>2672870</td>
<td>5</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>EnWikiSource.xml</td>
<td>1036.66</td>
<td>20</td>
<td>13423014</td>
<td>5</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>EnWikiVersity.xml</td>
<td>83.35</td>
<td>20</td>
<td>3333622</td>
<td>5</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>EnWikiTionary.xml</td>
<td>570.00</td>
<td>20</td>
<td>28656178</td>
<td>5</td>
<td>0.77</td>
</tr>
<tr>
<td>DBLP</td>
<td>DBLP.xml</td>
<td>130.72</td>
<td>32</td>
<td>4718588</td>
<td>5</td>
<td>0.58</td>
</tr>
<tr>
<td>U.S House</td>
<td>USHouse.xml</td>
<td>0.52</td>
<td>43</td>
<td>16963</td>
<td>16</td>
<td>0.77</td>
</tr>
<tr>
<td>SwissProt</td>
<td>SwissProt.xml</td>
<td>112.13</td>
<td>85</td>
<td>13917441</td>
<td>5</td>
<td>0.60</td>
</tr>
<tr>
<td>NASA</td>
<td>NASA.xml</td>
<td>24.45</td>
<td>61</td>
<td>2278447</td>
<td>8</td>
<td>0.66</td>
</tr>
<tr>
<td>Shakespeare</td>
<td>Shakespeare.xml</td>
<td>7.47</td>
<td>22</td>
<td>574156</td>
<td>7</td>
<td>0.64</td>
</tr>
<tr>
<td>Lineitem</td>
<td>Lineitem.xml</td>
<td>31.48</td>
<td>18</td>
<td>2045953</td>
<td>3</td>
<td>0.19</td>
</tr>
<tr>
<td>Mondial</td>
<td>Mondial.xml</td>
<td>1.75</td>
<td>23</td>
<td>147207</td>
<td>5</td>
<td>0.77</td>
</tr>
<tr>
<td>Baseball</td>
<td>Baseball.xml</td>
<td>0.65</td>
<td>46</td>
<td>57812</td>
<td>6</td>
<td>0.11</td>
</tr>
<tr>
<td>Treebank</td>
<td>Treebank.xml</td>
<td>84.06</td>
<td>250</td>
<td>10795711</td>
<td>36</td>
<td>0.70</td>
</tr>
<tr>
<td>Random</td>
<td>Random-R1.xml</td>
<td>14.20</td>
<td>100</td>
<td>1249997</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Random-R2.xml</td>
<td>53.90</td>
<td>200</td>
<td>3750002</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Random-R3.xml</td>
<td>97.85</td>
<td>300</td>
<td>7500017</td>
<td>30</td>
<td>0</td>
</tr>
</tbody>
</table>

6. XML testing corpus

6.1. Corpus characteristics

Determining the XML files that should be used for evaluating the set of XML compression tools is not a simple task. To provide an extensive set of experiments for assessing and evaluating the performance characteristics of the XML compression tools, we have collected and constructed a large corpus of XML documents. This corpus contains a wide variety of XML data sources and document sizes. Table 3 describes the characteristics of our corpus. Size denotes the disk space of XML file in MB. Tags represents the number of distinct tag names in each XML document. Nodes represents the total number of nodes in each XML data set. Depth is the length of the longest path in the data set. Data ratio represents the percentage of the size of data values with respect to the document size in each XML file. The documents are selected to cover a wide range of sizes where the smallest document is 0.5 MB and the biggest document is 1.3 GB. The documents of our corpus can be classified into four categories depending on their characteristics:

- **Structural documents**: this group of documents has no data contents at all. 100 % of each document size is preserved to store structure information. This category of documents is used as a control to assess the claim of XML conscious compressors on using the well-known structure of XML documents for achieving higher compression ratios on the structural parts of XML documents. Initially, our corpus consisted of 30 XML documents. Three of these documents were generated by using our own implemented Java-based random XML generator. This generator produces completely random XML documents to a parameterized arbitrary depth with only structural information (no data values). In addition, we created a structural copy for each document of the other 27 original documents – with data values – of the corpus. Thus, each structural copy captures the structure information of the associated XML original copy and removes all data values. In the rest of this paper, we refer to the documents which include the data values as original documents and refer to the documents with no data values as structural documents. As a result, the final status of our corpus consists of 57 documents, 27 original documents and 30 structural documents. The size of our own 3 randomly generated documents (R1, R2, R3) are indicated in Table 3 and the size of the structural copy of each original version of the document can be
computed using the following equation:

\[ \text{size(\text{structural})} = (1 - DR) \times \text{size(Original)} \]

where \( DR \) represents the data ratio of the document.

- **Textual documents**: this category of documents consists of a simple structure and a high ratio of its contents is preserved to the data values. The ratio of the data contents of these documents represent more than 70% of the document size.
- **Regular Documents**: this category of documents consists mainly of regular document structure and short data contents. This document category reflects the XML view of relational data. The data ratio of these documents is in the range of between 40 and 60 percent.
- **Irregular documents**: this category of documents consists of documents that have very deep, complex and irregular structure. Similar to purely structured documents, this document category is mainly focusing on evaluating the efficiency of compressing irregular structural information of XML documents.

6.2. Data sets

Our data set consists of the following documents:

- **EXI-Group** is a variant collection of XML documents included in the testing framework of the Efficient XML Interchange Working Group [4].
- **XMark-Group** the XMark documents model an auction database with deeply-nested elements. The XML document instances of the XMark benchmark are produced by the xmlgen tool of the XML benchmark project [15]. For our experiments, we generated three XML documents using three increasing scaling factors.
- **XBench-Group** presents a family of benchmarks that captures different XML application characteristics [13]. The databases it generates come with two main models: 1) Data-centric (DC) model contains data that are not originally stored in XML format such as e-commerce catalog data and transactional data 2) Text-centric (TC) model which represents text data that are more likely stored as XML. Each of these two models can be represented either in the form of a single document (SD) or multiple documents (MD). In short, these two levels of classifications are combined to generate four database instances: TCSD, DCSD, TCMD, DCMD. In addition, XBench can generate databases with 4 different sizes: small (11 MB), normal (108 MB), large (1 GB) and huge (10 GB). In our experiments, we only use TCSD and DCSD instances of the small and normal sizes.
- **Wikipedia-Group** Wikipedia offers free copies of all content to interested users [11]. For our corpus, we selected five samples of the XML dumps with different sizes and characteristics.
- **DBLP** presents the famous database of bibliographic information of computer science journals and conference proceedings.
- **U.S House** is a legislative document which provides information about the ongoing work of the U.S. House of Representatives.
- **SwissProt** is a protein sequence database which describes the DNA sequences. It provides a high level of annotations and a minimal level of redundancy.
- **NASA** is an astronomical database which is constructed by converting legacy flat-file formats into XML documents and then making them available to the public.
- **Shakespeare** represents the gathering of a collection of marked-up Shakespeare plays into a single XML file. It contains many long textual passages.
- **Lineitem** is an XML representation of the transactional relational database benchmark (TPC-H).
- **Mondial** provides the basic statistical information on countries of the world.
- **BaseBall** provides the complete baseball statistics of all players of each team that participated in the 1998 Major League.
- **Treebank** is a large collection of parsed English sentences from the Wall Street Journal. It has a very deep, non-regular and recursive structure.
- **Random-Group** is a group of documents that have been generated using our own implementation of a Java-based random XML generator. This generator is designed in a way to produce structural documents with very random, irregular and deep structures according to its input parameters for the number of unique tag names, maximum tree level and document size. We used this XML generator for producing three documents of different sizes and characteristics. The main aim of this group is to challenge the examined compressors and assess the efficiency of compressing the structural parts of XML documents.

7. Experiments

7.1. Testing environments

To ensure the consistency of the performance behaviors of the evaluated XML compressors, we ran our experiments on two different environments. One environment with high computing resources and the other with considerably limited
computing resources. Table 4 lists the setup details of our high resources environment and Table 5 lists the setup details of the limited one.

7.2. Examined compressors

In our study we considered, to the best of our knowledge, all XML compression tools which are fulfilling the following conditions:

1. Is publicly and freely available either in the form of open source codes or binary versions.
2. Is schema-independent. As previously mentioned, the set of compressors which is not fulfilling this condition is not commonly used in practice.
3. Be able to run under the version of the Linux version operating system that we use in our experiments.

In spite of the availability of source codes for the two compressors XAUST \[12\] and rngzip \[8\] have not been included in our study because they do not satisfy Condition 2. Similarly, the XGrind compressor \[14\] did not satisfy Condition 3. It always gives a fixed run-time error message during the execution process. The rest of the compressors do not satisfy Condition 1. The status of a lack of source code/binaries for a large number of the XML compressors proposed in literature, to the best of our search efforts and contact with the authors, and especially from the queriable class \[22,36,39,40,42,44\] was a bit disappointing for us. This has limited a subset of our initially planned experiments especially those which are targeted towards assessing the performance of evaluating the results of XML queries over the compressed representations.

As a result, we were only able to examine 9 XML compressors: 3 general purpose text compressors (gzip, bzip2, PPM) and 6 XML conscious compressors (XMill, XMLPPM, SCMPPM, XWRT, Exalt, AXECHOP). In the latest versions of the XMill source distribution, the intermediate binaries of the compressed format can be passed to one of three alternative back-end general purpose compressor: gzip, bzip2 and PPM. In our experiments we evaluated the performance of the three alternative back-ends independently. Hence, we refer to the three alternative back-ends with the names XMillGzip, XMillBzip and XMillPPM respectively.

7.3. Errors

During the run of our experiments, some tools failed to either compress or decompress some of the documents in our corpus. We consider the run as unsuccessful if the compressor fails to achieve either of the encoding and decoding processes of the test document. Thus, we had 57 runs for each compressor (one run per document). Fig. 1 presents the percentage of unsuccessful runs of each compressor. Table 6 presents a detailed list of the errors generated during our experiments. We have two main remarks about the results of Fig. 1:

- The general purpose text compressors have shown complete stability. They were able to successfully perform the complete set of runs. They are XML-Blind and thus would require no knowledge of the document-structure. Hence, they can deal with any XML document even if it suffers from any syntax or well-formedness problems. However, XML conscious
compressors are very sensitive to such problems. For example, some compressors which uses the Expat XML parser such as XMLPPM will fail to compress any XML document which uses external entity references if it does not have a dummy DTD declaration because the XML parser strictly applies the W3C specification and will consider this document as not well-formed.

– Except for the latest version of XMLPPM (0.98.3), none of the XML conscious compressors were able to execute the whole set of runs successfully. Moreover, AXECHOP and Exalt compressors have shown very poor stability. They failed to successfully complete the decoding parts in many of the runs. They were thus excluded from any consolidated results. Although an earlier version of XMLPPM (0.98.2) suffered from some problem in decompressing the Wikipedia data sets, the latest version of XMLPPM (0.98.3) released by Cheney during the time of doing the experiments of this work has fixed all earlier bugs and has shown to be the best XML conscious compressor from the stability point of view.

7.4. Performance metrics

We measure and compare the performance of the XML compression tools using the following metrics:

**Compression Ratio**: represents the ratio between the sizes of compressed and uncompressed XML documents as given by:

\[
\text{Compression Ratio} = \frac{\text{Compressed Size}}{\text{Uncompressed Size}}.
\]

**Compression Time**: represents the elapsed time during the compression process, i.e. the period of time between the start of program execution on a document until all the data are written to disk.

**Decompression Time**: represents the elapsed time during the decompression process i.e. the period of time between the start of program execution on reading the decompressed format of the XML document until delivering the original document.

For all metrics: the lower the metric value, the better the compressor.

7.5. Framework

We evaluated the performance characteristics of XML compressors by running them through an extensive set of experiments. The setup of our experimental framework was very challenging and complex. The details of this experimental framework is described as follows:

– We evaluated 11 XML compressors: 3 general purpose text compressors (gzip, bzip2, PPM) and 8 XML conscious compressors (XMLMillGzip, XMLMillBzip, XMLMillPPM, XMLPPM, SCMPPM, XWRT, Exalt, AXECHOP). For our main set of experiments, we evaluated the compressors under their default settings. The rational behind this is that the default settings are considered to be the recommended settings from the developers of each compressors and thus can be assumed as the best behavior. In addition to this main set of experiments, we run additional set of experiments with tuned parameters for the highest value of the level of compression parameter provided by some compressors (gzip, bzip2, PPM, XMLMillPPM, XWRT). That means in total we run 16 variant compressors. The experiments of the tuned version of XWRT could only be performed on the high resource setup because they require at least 1 GB of RAM.
Table 6
List of errors produced during the compression/decompression processes of the different compressors.

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Operation</th>
<th>Document Type</th>
<th>Document</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMill</td>
<td>C</td>
<td>Mondial.xml</td>
<td>Original</td>
<td>Parse error in line 13: Symbol '&gt;' expected after '/' in tag!</td>
</tr>
<tr>
<td>XWRT</td>
<td>D</td>
<td>EXI-factbook.xml</td>
<td>Structural</td>
<td>File corrupted (s.size()-WORD_MAX_SIZE)! Not enough memory!</td>
</tr>
<tr>
<td>XWRT</td>
<td>D</td>
<td>EXI-factbook.xml</td>
<td>Original</td>
<td>File corrupted (s.size()-WORD_MAX_SIZE)! Not enough memory!</td>
</tr>
<tr>
<td>SCMPPM</td>
<td>C</td>
<td>Mondial.xml</td>
<td>Original</td>
<td>Not well-formed document! &lt;DL&gt;</td>
</tr>
<tr>
<td>Exalt</td>
<td>C</td>
<td>DBLPxml</td>
<td>Structural</td>
<td>Size of the rule stack exceeded!</td>
</tr>
<tr>
<td>Exalt</td>
<td>C</td>
<td>DBLPxml</td>
<td>Structural</td>
<td>Size of the rule stack exceeded!</td>
</tr>
<tr>
<td>Exachop</td>
<td>C</td>
<td>DBLPxml</td>
<td>Structural</td>
<td>Unexpected Exception</td>
</tr>
<tr>
<td>Exachop</td>
<td>D</td>
<td>EnWikiNew.xml</td>
<td>Structural</td>
<td>terminate called after throwing an instance of 'std::bad_alloc'</td>
</tr>
<tr>
<td>Exachop</td>
<td>D</td>
<td>EXI-Array</td>
<td>Structural</td>
<td>in processEndElement() trying to read from empty path stack</td>
</tr>
<tr>
<td>Exachop</td>
<td>C</td>
<td>DBLPxml</td>
<td>Original</td>
<td>Fatal Error</td>
</tr>
<tr>
<td>Exachop</td>
<td>C</td>
<td>EnWikiQuote.xml</td>
<td>Original</td>
<td>Command terminated by signal 11</td>
</tr>
<tr>
<td>Exachop</td>
<td>C</td>
<td>EXI-factbook.xml</td>
<td>Original</td>
<td>Command terminated by signal 11</td>
</tr>
<tr>
<td>Exachop</td>
<td>D</td>
<td>baseball.xml</td>
<td>Original</td>
<td>Command terminated by signal 11</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 6 (continued)

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Operation</th>
<th>Document</th>
<th>Type</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axechop</td>
<td>D</td>
<td>EnWikiNew.xml</td>
<td>Structural</td>
<td>terminate called after throwing an instance of &quot;std::bad_alloc&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mondial.xml</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nasa.xml</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shakespeare.xml</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XBench-DCSD-Normal.xml</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XBench-DCSD-Small.xml</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XBench-TCSD-Normal.xml</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XBench-TCSD-Small.xml</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Our corpus consists of 57 documents: 27 original documents, 27 structural copies and 3 randomly generated structural documents (see Section 6.1).
- We run the experiments on two different platforms. One with limited computing resources and the other with high computing resources.
- For each combination of an XML test document and an XML compressor, we run two different operations (compression–decompression).
- To ensure accuracy, all reported numbers for our time metrics (compression time–decompression time) (see Section 7.4) are the average of five executions with the highest and the lowest values removed.

The above details lead to the conclusion that our number of runs was equal to 9120 on each experimental platform \((16 \times 57 + 2 \times 5)\), i.e. 18240 runs in total.

In addition to running this huge set of experiments, we needed to find the best way to collect, analyze and present this huge amount of experimental results. To tackle this challenge, we created our own mix of Unix shell and Perl scripts to run and collect the results of these huge number of runs. In this paper, we present the most important parts from results of our experiments. For full detailed results, we refer the reader to the web page of this experimental study [1].

7.6. Experimental results

In this section we report the results obtained by running our exhaustive set of experiments. Figs. 9 to 11 represent an important part of the results of our experiments. Several remarks and guidelines can be observed from the results of our exhaustive set of experiments. Some key remarks are given as follows:

- Fig. 7(a) shows that the three alternative back-ends of XMill compressor achieve the best compression ratio over the structural documents. Fig. 2(a) shows that XMillPPM achieves the best compression ratio for all the datasets. The irregular structural documents (Treebank, R1, R2, R3) are very challenging to the set of the compressors. This explains why they all had the worst compression ratios.
- Figs. 2(b) and 7 show that gzip-based compressors (gzip, XMLGzip) have the worst compression ratios. Excluding these two compressors, Fig. 7 shows that the differences on the average compression ratios between the rest of compressors are very narrow. They are very close to each other, the difference between the best and the worst average compression ratios is less than 5%. Among all compressors, SCMPMM achieves the best average compression ratio.
- Figs. 3, 4, 5, 6 show that the gzip-based compressors have the best performance in terms of compression time and decompression time metrics on both testing environments. The compression and decompression times of the PPM-Based compression scheme (XMillPPM, XMLPPM, SCMPMM) are much slower than the other compressors. Among all compressors, SCMPMM has the longest compression and decompression times.
- Fig. 8 illustrates the overall performance of XML compressors on the high and limited resources setup where the values of the performance metrics are normalized with respect to bzip2. The results of this figure illustrate the narrow differences between the XML compressors in terms of their compression ratios and the wide differences in terms of their compression and decompression times.
- The results of Figs. 9 and 10 show that the tuned run of XWRT with the highest level of compression ratio achieves the overall best average compression ratio with very expensive cost in terms of compression and decompression times.

8. Ranking

Obviously, it is a nice idea to use the results of our experiments and our performance metrics to provide a global ranking of XML compression tools. This is however an especially hard task. In fact, the results of our experiments have not shown a clear winner. Hence, different ranking methods and different weights for the factors could be used for this task. Deciding the weight of each metric is mainly dependant on the scenarios and requirements of the applications where these compression tools could be used. In this paper we used three ranking functions which give different weights for our performance metrics.
These three rankings function are defined as follows:

- \( WF_1 = \left( \frac{1}{3} \times CR \right) + \left( \frac{1}{3} \times CT \right) + \left( \frac{1}{3} \times DCT \right) \);
- \( WF_2 = \left( \frac{1}{2} \times CR \right) + \left( \frac{1}{4} \times CT \right) + \left( \frac{1}{4} \times DCT \right) \);
- \( WF_3 = \left( \frac{3}{5} \times CR \right) + \left( \frac{1}{5} \times CT \right) + \left( \frac{1}{5} \times DCT \right) \),

where \( CR \) represents the compression ratio metric, \( CT \) represents the compression time metric and \( DCT \) represents the decompression time metric. In these ranking functions we used increasing weights for the compression ratio (CR) metric (33%, 50% and 60%) while \( CT \) and \( DCT \) were equally sharing the remaining weight percentage for each function. Fig. 11 shows that gzip and XMLGzip are ranked as the best compressors using the three ranking functions and on both of the testing environments. In addition, Fig. 11 illustrates that none of the XML compression tools has shown a significant or noticeable improvement with respect to the compression ratio metric. The increasing assignment for the weight of CR do not change the order of the global ranking between the three ranking functions.
Fig. 3. Detailed compression times on the high resources setup. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 4. Detailed compression times on the limited resources setup. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

9. Conclusions

In this paper we surveyed the state-of-the-art of XML compression techniques. We presented an extensive experimental study of the available implementations of these techniques. We reported the behavior of nine XML compressors using a large corpus of XML documents which covers the different natures and scales of XML documents. We believe that this paper could be valuable for both the developers of new XML compression tools and interested users as well. For developers, they can use the results of this paper to effectively decide on the points which can be improved in order to make an effective contribution. For this category of readers, we recommend tackling the area of developing stable and efficient queriable XML compressors. Although there has been a lot of literature presented in this domain, our experience from this study
lead us to the result that we are still missing efficient, scalable and stable implementations in this domain. For users, this study could be helpful for making an effective decision to select the most suitable compressor for their requirements. For example, for users with the highest compression ratio requirement, the results of Fig. 9 recommends the use of either the PPM compressor with the highest level of compression parameter (ppmd e-o16 document.xml) or the XWRT compressor with the highest level of compression parameter (xwrt -i14 document.xml) (if they have more than 1 GB RAM on their systems) while for the users with the fastest compression time and moderate compression ratio requirements, gzip and XMILLGzip are considered to be the best choice (Fig. 11).

From the experience and the results of this experimental study, we can draw the following conclusions and recommendations:
The primary innovation in the XML compression mechanisms was presented in the first implementation in this domain by XMill. It introduced the idea of separating the structural part of the XML document from the data part and then groups the related data items into homogenous containers that can be compressed separately. This separation improves the further steps of compressing these homogenous containers using the general purpose compressors or any other compression mechanism because they can detect the redundant data more easily. Most of the following XML compressors have simulated this idea in different ways.

Schema-dependent compressors are not preferable or commonly used in practice because there is no guarantee that the schema information of the XML documents is always available and in the required format (DTD, XML Schema, RELaxNG).

The dominant practice in most of the XML compressors is to utilize the well-known structure of XML documents for applying a pre-processing encoding step and then forwarding the results of this step to general purpose compressors. Consequently, the compression ratio of most XML conscious compressors is very dependent and related on the general purpose compressors such as gzip, bzip2 or PPM. Fig. 7 shows that none of the XML conscious compressors has achieved an outstanding compression ratio over its back-end general purpose compressor. The improvements are always not significant with 5% being the best of cases. This fact could explain why XML conscious compressors are not widely used in practice.
Fig. 8. Overall performance of compressing original documents. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 9. Average Compression Ratios of original documents (Tuned Parameters).
Fig. 10. Average compression and decompression times over original documents on high resources setup (Tuned Parameters). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 11. Ranking Functions of compressing original documents. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
– The compression time and decompression time metrics play a crucial role in the ranking of XML compressors.
– The authors of the XML compression tools should provide more attention to provide the source code of their implementations available. Many tools presented in the literature – specially the queriable ones – have no available source code which prevents the possibility of ensuring the repeatability of the reported numbers. It also hinders the possibility of performing fair and consistent comparisons between the different approaches. For example in [39], the authors compared the results of their implementation Xseq with XBzip using an inconsistent way. They used the reported query evaluation time of XBzip in [27] to compare with their times although each of the implementation is running on a different environment.
– There are no publicly available solid implementations for grammar-based XML compression techniques and queriable XML compressors. These two areas provide many interesting avenues for further research and development.

As a future work, we are planning to continue maintaining and updating the web page of this study with further evaluations of any new evolving implementations of XML compressors. In addition, we will enable the visitor of our web page to perform their online experiments using the set of the available compressors and their own XML documents.

Supplementary material

The online version of this article contains additional supplementary material. Please visit doi:10.1016/j.jcss.2009.01.004.

References