TREECHOP: A TREE-BASED QUERY-ABLE COMPRESSOR FOR XML

Gregory Leighton, Tomasz Müldner, James Diamond
{005985L, tomasz.muldner, james.diamond}@acadiau.ca
Jodrey School of Computer Science, Acadia University, Wolfville, Nova Scotia, Canada

ABSTRACT

XML is a popular meta-language that facilitates the interchange and access of data. However, XML's verbose nature may increase the size of a data set as much as ten-fold. In this paper, we present a novel technique for lossless XML compression, called TREECHOP, which supports querying of compressed XML data without requiring full decompression. Unlike other query-capable XML compression schemes, TREECHOP requires only a single pass over the input document during the compression process, resulting in an efficient, online operation that is well-suited for transmission of compressed XML documents over a network.

1 INTRODUCTION

The eXtensible Markup Language (XML) [1] is a World Wide Web Consortium (W3C) endorsed standard for semi-structured data. It allows data to be surrounded by textual markup (*elements* and *attributes*) that serves to describe its semantics. The inclusion of structural information with the data grants XML great flexibility, at the cost of increased verbosity. It is not uncommon for the XML representation of a set of data to be as much as ten times as large as alternative representations (e.g. data in comma-separated value format).

In recent years, messaging has been one of the most common applications of XML. One example is the Web Services initiative, in which network services can be discovered, described, and invoked in a platform- and implementation-independent way via the exchange of XML messages. Such applications would benefit from a compression scheme which operates online and allows queries to be carried out directly on compressed data. In this paper, we present TREECHOP, an XML-conscious compression scheme which achieves both of these objectives.

1.1 Related Work

XMill [2] represents the pioneering work in the area of XML-conscious compression. Its compression strategy separates the structural information of an XML document from the contained data. Data values are then grouped in containers according to the identity of the enclosing element or attribute, and gzip [3] is subsequently applied to each individual container. A container expression language allows the user to substitute alternative compression strategies for gzip. Although XMill often outperforms gzip on XML data, the original structure of the document is disrupted during the compression process, which precludes online processing. This serves to limit the usefulness of XMill for XML messaging applications. In addition, the XMill encoding format does not allow querying of compressed data.

XMLPPM [4] achieves a higher degree of compression via the use of multiplexed hierarchical models and the PPM [5] text compression method. As with XMill, compressed data cannot be queried.

XGRIND [6] was the first XML-conscious compression scheme to support querying without full decompression. Element and attribute names are encoded using a byte-based scheme, and character data is compressed using non-adaptive Huffman coding [7]. Use of the latter technique significantly slows down the compression process, since two passes over the original document are required (first to gather probability data, and a second time to perform the encoding).

XPRESS [8] also supports querying of compressed data and claims to achieve better compression than XGRIND. However, it uses a semi-adaptive form of arithmetic coding which also necessitates two passes over the original XML document.

1.2 Contributions

This paper presents linear time algorithms for compressing, decompressing, and querying XML data. Unlike the query-able XML compression schemes described in [6] and [8], compression requires only a single pass through the input XML document.

1.3 Organization

Section 2 of this paper describes the compression, decompression, and querying strategies used in TREECHOP. Experimental results comparing the compression and speed of TREECHOP versus alternative compression routines are presented in Section 3. Section 4 concludes the paper.

2 TREECHOP

In this section, we define some notational conventions used in the subsequent discussions on the compression, decompression, and querying strategies employed in TREECHOP. We begin by describing the XML document tree.

This work has been partially supported by NSERC grants 41-0-235041 and 41-0-235201

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- start of PO -->
  <PurchaseOrder no="1456">
    <Date>06/05/05</Date>
    <CustomerID>765345</CustomerID>
    <Order>
      <Item>
        <ProductNo>P-4534</ProductNo>
        <Quantity>2</Quantity>
      </Item>
      <Item>
        <ProductNo>P-9182</ProductNo>
        <Quantity>1</Quantity>
      </Item>
    </Order>
  </PurchaseOrder>
  <!-- end of PO -->
```

Fig. 1. Example XML document

The root node of the document tree corresponds to the root element in the XML document. Any information appearing before the root element (such as the XML declaration, DOCTYPE declaration, processing instructions, or comments) is stored in a data container called the *prologue*. Similarly, any comments or processing instructions occurring after the end tag of the root element are stored in a data container called the *epilogue*.

Character data (such as attribute values and text occurring between an XML element's beginning and ending tags) are leaf nodes in the tree. All other data types appear as non-leaf nodes. There are five non-leaf node types, described below.

- **attribute node:** this subtype corresponds to the occurrence of an attribute node within the source XML document. Each attribute has a single child, a leaf node containing the attribute's data value.
- **CDATA node:** represents a CDATA section within the source XML document. The contents of the CDATA section are contained in the label for the node.
- **comment node:** corresponds to a comment occurring between the start and end tags of the XML document's root element. The text between the delimiting <!-- and --> comment markers is stored in the node's label.
- **element node:** represents an occurrence of an XML element within the document. Each element node may have multiple children, including nodes representing nested elements, attributes, comments, or processing instructions. Non-empty element nodes have a leaf node child containing the character data enclosed by the element's start and end tag. The root node in the document tree is always an instance of this type.
- **processing instruction node:** each occurrence of this subtype corresponds to the appearance of a processing instruction in the source XML document. Any text between the delimiting <? and ?> markers forms the node's label.

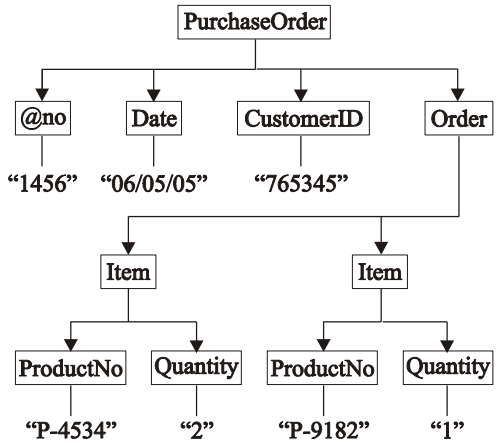

Fig. 2. Tree Representation of the XML Document in Fig. 1

Fig. 1 depicts an XML document and Fig. 2 shows the equivalent document tree representation.

Definition 1: Each node in the XML document tree possesses a textual *label*. In the case of XML elements, the label is the name of the element; for XML attributes, the label is formed by concatenating '@' with the name of the attribute. In the case of comments, processing instructions, and CDATA sections, the label consists of all text between the delimiting section markers.

As an example, the root element of the document tree in Fig. 2 has the label *PurchaseOrder*, while the *no* attribute associated with the *PurchaseOrder* element is assigned the label *@no*.

Definition 2: The *path* of a non-leaf node v_n in the XML document tree is a sequence $/L_1/L_2/\ldots/L_n$ of one or more '/'-separated labels that traces a route from the root node v_1 to v_n, where L_i is the label of node v_i.

In the document tree depicted in Fig. 2, each of the nodes labeled *Quantity* is assigned the same path */PurchaseOrder/Order/Item/Quantity*, while the node labeled *@no* has the path */PurchaseOrder/@no*.

2.1 Compression Strategy

The compression process in TREECHOP begins by conducting a SAX-based [9] parsing of the XML document. As tokens are returned by the parser, new tree nodes are created and then written out to the compression stream in depth-first order. This approach avoids building an in-memory representation of the entire document tree.

Each non-leaf node is assigned a binary codeword. This codeword is uniquely assigned based on the path of the tree node. If there are multiple nodes with the same absolute path, each occurrence will receive the same codeword. For example, in the tree shown in Fig. 2, each of the two instances of */PurchaseOrder/Order/Item* will be assigned the same codeword.

The codeword $C(v)$ assigned to a non-leaf node v with parent node p is formed by the concatenation of three codes $C(p)$, $G(v)$, and $T(v)$, where

Node Type	$T(v)$
Element	000
Attribute	001
Comment	010
CDATA	011
Processing Instruction	100

Table 1. Values for $T(v)$ by node type

- $C(p)$ represents the codeword assigned to p
- $G(v)$ is a Golomb code [10] assigned to v based on its ordering relative to p. If v is the n-th distinct child node of p (where two nodes are said to be distinct if they have different paths), then we form $G(v)$ by concatenating the unary code for $q + 1$ with the binary code for r, where

$$q = \lfloor (n-1)/3 \rfloor \quad (1)$$

$$r = n - 3q - 1 \quad (2)$$

  The constant three in (1) and (2) is a parameter of Golomb coding; the reasons for this particular choice can be found in [11].
- $T(v)$ is a sequence of 3 bits used to indicate the node type. Table 1 lists the $T(v)$ values for each node type.

The codeword of the root consists of 00000. An example of the codeword assignment scheme is provided in Table 2, pertaining to the document tree depicted in Fig. 2.

This encoding scheme has three important properties: (1) the codeword for each node is prefixed by its parent's codeword; (2) two nodes share the same codeword if and only if they have the same path; and (3) the structure of the original XML document is maintained by the encoding scheme.

The encoding information for each tree node is written to the encoding stream in an adaptive fashion. Each non-leaf node is encoded as a 3-tuple *(L, C, D)*, where *L* is a byte indicating the bit length of the codeword; *C* is the codeword assigned to the node, consisting of a sequence of $\lceil L / 8 \rceil$ bytes; and *D* is the textual data stored in the node. A reserved byte value is used to indicate to the decoder that raw character data is forthcoming in the encoding stream; once *D* has been transmitted, a second reserved byte value is used to signal the end of the character data sequence.

Leaf nodes are transmitted in the same manner as *D*, described above. For the second and subsequent occurrences of a particular codeword, only the 2-tuple *(L, C)* is transmitted since the decompressor is able to infer the value of *D* at that point.

Information about a node *N* is written to the encoding stream immediately after *N* is assigned a codeword. This allows the decoder to set about decoding *N* before encoding of other nodes has been received. As node information is added to the compression stream, it is compressed using gzip.

2.2 Decompression Strategy

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 incrementally. A code table is used to store $(L, C) \rightarrow D$ mappings for previously-encountered tree nodes. In addition, a stack is employed to maintain proper nesting of elements during the decompression process.

Node Path	$C(v)$
/PurchaseOrder	00000
/PurchaseOrder/@no	0000000001
/PurchaseOrder/Date	00000010000
/PurchaseOrder/CustomerID	00000011000
/PurchaseOrder/Order	00000100000
/PurchaseOrder/Order/Item	0000010000000000
/PurchaseOrder/Order/Item/ProductNo	00000100000000000000000
/PurchaseOrder/Order/Item/Quantity	00000100000000000010000

Table 2. Assigned codewords for the document tree in Fig. 2

As each non-leaf tree node is encountered in the compression stream, the decompressor determines the node type by examining the final three bits in the codeword. The type information is then used to surround the *D* value for this node with the appropriate XML syntax before emitting it to the decompression stream.

2.3 Querying Strategy

Exact-match queries can be carried out via a single scan through the compression stream. The query processor employs a stack to keep track of the current path; when the query predicate path is first encountered, the associated codeword *C* is recorded and the next occurring *D* value is extracted from the compression stream as a query match. Subsequently, the remainder of the stream is scanned for future occurrences of *C*. With each match, the associated *D* value is extracted from the stream.

Range queries are handled in a similar manner, except that each query match additionally requires that the corresponding *D* value be converted into a numeric value and tested to see if it falls within the query range before it is returned as a search result.

3 EXPERIMENTAL RESULTS

This section presents the results of two sets of experiments that were carried out to evaluate the effectiveness of TREECHOP in compressing and transmitting XML data over a TCP/IP network.

3.1 Compression Rates

Fig. 3 illustrates the compression rates achieved by TREECHOP, gzip, and XGRIND on four XML files. Columns A, B, C, and D respectively indicate performance on a file containing player statistics from the 1998 Major League Baseball season, Shakespeare's play *Macbeth*, a highly-structured document containing 150 employee records, and a similar document with 100000 employee records. Table 3 describes the structural characteristics of each test file, including the original document size, the number of elements and attributes, and the total number of characters in the data sections. The results indicate that XGRIND performs significantly worse than either gzip or TREECHOP on each file, while TREECHOP slightly outperforms gzip in all cases.

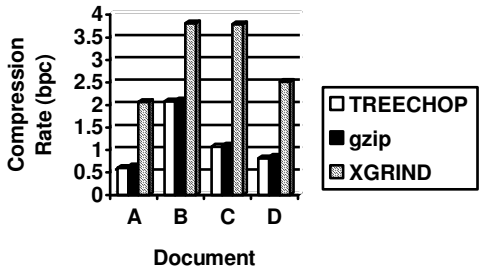

Fig. 3. Compression performance of TREECHOP, gzip, and XGRIND (measured in bits-per-character).

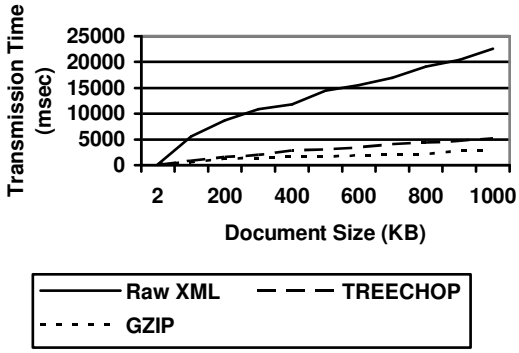

Fig. 4. Transmission speed of TREECHOP, gzip, and raw XML data over a TCP/IP network.

File	Size(KB)	Elements	Attributes	Data
baseball	788	27080	0	230970
macbeth	175	3975	0	97625
150emp	26	901	150	8277
100000emp	16831	600001	100000	5534311

Table 3. Characteristics of XML documents used in the compression experiment.

3.2 Compression/Decompression Speed

To evaluate the speed of TREECHOP's compression and decompression strategies, an experiment was carried out in which a set of documents ranging in size from 2 KB to 1 MB were compressed, transmitted over a TCP socket connection to a remote server located 20km (12 miles) from the client, and decompressed on the server side to reproduce the original document. Each document consisted of a set of employee records, similar to test cases C and D in Section 3.1. The performance of TREECHOP was compared with gzip and with uncompressed transmission of each document. Results for XGRIND were not included since the current implementation does not support online transmission of compressed data between networked systems.

The results of this experiment are depicted in Fig. 4. Not surprisingly, both gzip and TREECHOP experience a widening performance advantage over raw XML data transmission as the document size increases. In addition, gzip performs slightly faster than TREECHOP on the larger documents in the test set (due to the additional computational expense of calculating and decoding the codeword for each tree node).

When interpreting the results, it is worth noting that as the physical distance between the client and server systems is increased, the slight speed advantage of gzip during the compression and decompression phases may eventually be eclipsed by the sheer expense of sending data across the network (if the network approaches its saturation point). When this is the case, TREECHOP's ability to compress data at a better rate than gzip will allow it to achieve superior transmission rates. Additionally, in cases where the receiving server program is only interested in a subset of the document content (e.g. the name of each employee), it would be more efficient to perform a search on TREECHOP-compressed data in lieu of carrying out a full decompression of the document.

4 CONCLUSIONS

An extended version of this paper is available in [11].

REFERENCES

[1] Extensible Markup Language (XML) 1.0 (3rd ed.). http://www.w3.org/TR/REC-xml.

[2] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," in *2000 Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pp. 153-64.

[3] gzip, http://www.gzip.org.

[4] J. Cheney, "Compressing XML with multiplexed hierarchical PPM models," in *2001 Proc. IEEE Data Compression Conference*, pp. 163-72.

[5] J.G. Cleary and I.H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Comm.*, vol. 32, no. 4, pp. 396-402, Apr. 1984.

[6] P.M. Tolani and J.R. Haritsa, "XGRIND: a query-friendly XML compressor," in *Proc. 2002 Int'l Conf. on Database Eng.*, pp. 225-34.

[7] D. Huffman, "A method for the construction of minimum redundancy codes," in *Proc. of the IRE.*, vol. 40, no. 9, pp. 1098-1101.

[8] J-K. Min, M-J. Park and C-W. Chung, "XPRESS: a queriable compression for XML data," in *Proc. 2003 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 122-33.

[9] Simple API for XML (SAX), http://www.saxproject.org.

[10] S.W. Golomb, "Run-length encodings," *IEEE Trans. on Info. Theory* IT-12(3), pp. 399-401, July 1966.

[11] G. Leighton, T. Müldner, J. Diamond. "TREECHOP: A tree-based query-able compressor for XML (extended version)", http://cs.acadiau.ca/technicalReports/.