Fast Adaptive Data Compression for Information Retrieval

Michael J. Nelson
School of Library and Information Science
University of Western Ontario
London, Ontario N6G 1H1
CANADA

1 Introduction

Despite storage devices such as optical disks with very large storage capacities, there is a very active interest in data compression. The advantages of storing even more information is very attractive. For CD-ROMs with a fixed standard storage capacity, sometimes compression is needed for large databases and all the associated files and indexes to be stored on one CD-ROM.

One method of characterizing compression methods is by type of strings represented from the original text and type of codes used to represent these strings in the compressed version. Both the original and codes can be either fixed or variable. An example of the fixed original to fixed codes is a type bigram encoding where certain frequently occurring pairs of letters are replaced a code which has the same length as one letter. Classic Huffman coding is an example of the fixed length original text and variable length codes. Every character has a code assigned based on the frequency of occurrence in the text, the most frequent characters having the shortest codes. An example of variable to fixed are various dictionary based methods which represent each entry in the dictionary by a fixed length code and the entries in the dictionary are text fragments of varying length. Finally, an example of the variable to variable length code are the adaptable methods explained below.

Another method of classification is static versus adaptive. Static methods devise a coding scheme which does not change over the text in the document. This is usually based on a complete scan of the data to be compressed, or at least a large sample. If the data is relatively homogeneous, for example English text, a sample of another text may also be used. Static methods have been the most popular in information retrieval since in most cases the data itself is relatively static and can be completely analyzed before storage. Adaptive methods depend on analyzing the incoming text as it is processed, thus changing the codes as different patterns of data are found. The main advantage is the adaptability when the data being compressed changes its basic nature. Also there is no extra storage for the coding scheme which is dynamically built as the compression is done. So for archiving a large number of different types of files this adaptable methods have become the norm.

Adaptable methods are mostly based on Ziv and Lempel [1977] and Ziv and Lempel [1978] (referred to as the LZ77 and LZ78 methods). The LZ77 family of algorithms codes the text as either a direct copy of the text or as a pointer to previously processed text where the same string of characters occurred. The LZ78 family codes the text as a pointer to a dynamic dictionary kept in memory. There are many variations and implementations of both types described in the literature. For example the UNIX compress command uses an algorithm from the LZ78 family as described by Welch [1984] and denoted as LZW. A good review of these algorithms can be found in the book by Bell [1990].

2 Compression Problem in Information Retrieval

2.1 Characteristics

Most compression methods in the literature are good for archival applications. This means the whole file is compressed into a single output file and the whole compressed file is decompressed when needed. To display the results of a query in information retrieval applications a small set of records from the database must be retrieved. Thus single records or small blocks of data need to be decompressed during interactive sessions. The decompression method should be fast and use as little memory as possible. On the other hand the compression method can be slower as in most information retrieval applications the database does not change very quickly. Because of these characteristics most research has been on using static coding methods.

2.2 Recent Compression Methods in Information Retrieval

Klein, Bookstein and Deerswater [1989] used Huffman codes on single characters and bigrams (pairs of characters) to obtain a percent remaining figure of 65% for a database on a CD-ROM. Bookstein, Klein and Ziff [1991] extended this by applying a Markov model of occurrence and reduced the database from 595.4 MB to 215.8 MB (36.2% of the original). This needed large amounts of computation on the original database to find the best model. This extra computation may be justified for a static database stored on CD-ROM.

Impressive results have also been achieved by Liu and Yu [1991] at the expense of using large amounts of memory during both compression and decompression. The basic idea was to use a Huffman code on words instead of single characters as the original methods did. This means keeping a large dictionary of the most frequent words in memory. On a 128.8 MB file they obtained a figure of 35.19% remaining while using a maximum of 640 kB of memory for the dictionary. The algorithm also ran slower than the UNIX utilities for compression.

Note that both these methods are static methods. The advantage of these methods is that the characteristics of the whole database can be used to compress each block or record. Also, in contrast to adaptive methods, there is no start up time to adapt to

the data when retrieving single records or blocks. On the other hand a good adaptive algorithm will dispense with the pre-processing and will not need extra storage for the coding scheme or dictionary. An adaptive algorithm which can be used for information retrieval is described in the next section.

3 The LZRW1 Algorithm.

3.1 Description

Williams [1991] describes an extremely fast compression algorithm which runs in linear time and has a good worst case running time which he calls LZRW1. In addition it does not require any extra memory for dictionaries or tables during decompression and works well for small blocks of data as it adapts quickly. These characteristics make this algorithm an ideal candidate for information retrieval. The information retrieval system can decompress records (or blocks) as needed for display with very little overhead.

The algorithm is based on LZ77, which converts text into a sequence of items, each of which is a literal byte or code representing a copy of a previously sent string of bytes. LZRW1 uses a single bit to indicate a literal or copy item. These are grouped into 16-bit words to control 16 items. If the bit is '0' then the next byte is a literal. If the bit is a '1' then the item is a two-byte code representing the length in 4 bits (from 3 to 15) and the offset of the text to copy in 12 bits (from 1 to 4095). This means the maximum match length of text is eighteen and the farthest back we can point is 4095 bytes.

To search the text already processed a hash table of 4096 pointers is maintained during compression. The pointers point to an occurrence of a sequence of 3 bytes (a 2 byte sequence will not compress). The hash table is updated after every item is coded.

This makes the decompression algorithm very simple and extremely fast. The program checks the indicator bit, if it is "0" transfer one byte to the output otherwise use the offset and the length stored in the code are used to copy several bytes from the buffer to the output.

An exact description of the algorithm along with C programs is included in the original paper by Williams [1991]. This paper and other materials including further extensions of the basic algorithm called LZRW2 to LZRW5 are available by anonymous FTP from sirius.itd.adelaide.edu.au (129.127.40.3) in directory ~pub/misc.

3.2 Basic Performance

The algorithm gave an average compression of 55.5% (ratio of compressed to not compressed) on the corpus of files used by Bell [1990]. For comparison this is about 10% worse than the LZW algorithm used in UNIX compress, but of course is much faster.

Fiala and Greene [1989] also present some data on several compression methods based on the LZ77 method. They state:

> "The 4,096 window of A1 is roughly optimal for fixed size copy and literal codewords... To exploit a larger window we must use a variable-width encoding..."

We want to avoid variable width codes to improve speed. Another consideration is that these results assume that this is a sliding window onto a much larger file and that the whole file is processed sequentially. In our application we process one record or one block of data at a time.

Since our interest is in decompressing relatively small blocks of data, two sample databases were set up to test the compression ratio for relatively small blocks of data. The first is a set of 135 records from online searches of Dissertation Abstracts which include a bibliographic reference and a long abstract. The average record size is 2293 bytes. Secondly, a set of short bibliographic records from the catalog at the School of Library and Information Science were copied to a simple ASCII file.

The performance of the LZRW1 algorithm as it progresses through a block of data is shown in Figure 1. The percent remaining was sampled every 50 bytes and plotted on the graph. This was done for twenty 16k blocks and averaged. Note that at the beginning each block the space needed actually expands as the indictor bits are needed. The percent remaining only changes one or two percent between 8k and 16k. Figure 2 shows the percent remaining for individual records using a modified algorithm described in section 4.3.2. For comparison a basic Huffman coding on the dissertations database gave 63.7% remaining.

Both the compression and decompression are very fast compared to most other algorithms. The decompression is extremely simple and fast which means very little overhead in programming, memory or processing time. Williams [1991] gives average performance of 49kB per second for compression and 151kB per second for decompression when written in assembly language on and 8MHz Macintosh-SE computer. Implemented in C on a UNIX machine it runs about four time faster than the UNIX Compress program.

3.3 Improvements to LZRW1

3.3.1 Changing the length code

The first change is relatively minor. Instead of coding the length of match from 3 to 16, use 3 to 18 and subtract 3 before coding. This gives about a 1% improvement. This is documented by Williams as algorithm LZRW1-A.

3.3.2 Using 7-bit.

In many applications the data can be represented by the basic character set of 128 characters in 7-bit ASCII. If there are only a few occurrences of 8-bit characters, one of the 7-bit codes could be used as a shift code to indicate that the next byte is an 8-bit character. Then the first bit of each character can be used for the flag to indicate if a character is being sent or a code is being sent. This has the potential saving of 12.5% (1 in eight bits), but we must reduce the pointer to previous text from 12 bits (0 to 4095) to 11 bits (0 to 2047). For small blocks this will not be a problem. In table 1 we can see the improvement for small

blocks but there is actually a slight increase in compressed size for blocks of 8192 characters.

Table 1.

Block size\database	8-bit characters	7-bit characters
1024	72.8 %	65.2 %
2048	66.0 %	60.7 %
4096	61.2 %	58.9 %
8192	59.3 %	57.9 %

3.3.3 Pre-loaded Buffer

Another way to make adaptive methods adapt faster from the beginning of a block is to use a buffer of standard text in the compressor to start with which has the effect of making the text stream seem longer to the compression program. This makes the compressor use a little more memory and run slightly slower. The decompressor must also have a larger buffer and keep a copy of the hash table for the pre-loaded text to start the decompression of each block.

The ideal pre-loaded buffer would contain the most frequent strings copied in the LZ77 methods to get it started fast. In addition the size of the buffer and the optimum order of the strings in the buffer must be determined. This seems like a difficult computational problem. This also means pre-processing all the data, which negates one of the big advantages of the adaptive method. Instead there are several ad-hoc methods which could be used: standard phrase lists from language studies, stop word lists, and random sample of a few blocks of data.

A few tests were run for an even simpler method where the first block in the file is the pre-loaded buffer. Results are summarized in Tables 2 and 3. The advantage of a small pre-loaded buffer makes the effect of block size minimal. The optimum size seems to be a 1024 byte pre-loaded buffer which then masks the effect of increasing the block size to get better compression.

Table 2. % remaining in Dissertations database

	Size of pre-loaded Buffer			
Block Size	0	512	1024	2048
1024	65.26	58.67	57.21	58.47
2048	60.67	57.77	57.01	57.77
4096	58.95	57.30	57.07	57.29

Table 3. % remaining in Cataloguing database

	Size of pre-loaded Buffer			
Block Size	0	512	1024	2048
1024	51.77	46.86	44.95	47.06
2048	47.09	45.03	44.18	45.20
4096	45.15	44.14	43.82	44.25

4 Summary

Fast adaptive compression methods can be used for small blocks of data in information retrieval with results at least as good as Huffman coding. Both the compression and decompression methods are easy to program and are easy to optimize to make the performance penalty of compression negligible. In addition it is easy to add or modify records and if the nature of the records changes over time the method will adapt. Another possible application is for databases with a mixture of record types as this algorithm works for binary data such as pictures or sound. For binary data using the pre-loaded buffer may not be as effective.

BIBLIOGRAPHY

Bell, T. C., Cleary, J. G., & Witten, I. H. (1990). *Text compression*. Englewood Cliffs, New Jersey: Prentice Hall.

Fiala, E. R., & Greene, D. H. (April 1989). Data compression with finite windows. *Communications of the ACM, 32*(4), 490-505.

Williams, R. N. (1991). *Adaptive data compression*. Boston: Kluwer Academic.

Williams, R. N. (1991). An extremely fast Ziv-Lempel data compression algorithm. In J. A. Storer, & J. H. Reif (Eds.), *IEEE Computer Society Data Compression Conference, Snowbird, Utah*.IEEE.

Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory, 24*(5), 530-536.

Ziv, J., & Lempel, A. (May 1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory, 23*(3), 337-343.

Fig. 1. Instataneous Compression
16k Blocks (LZRW1-a)

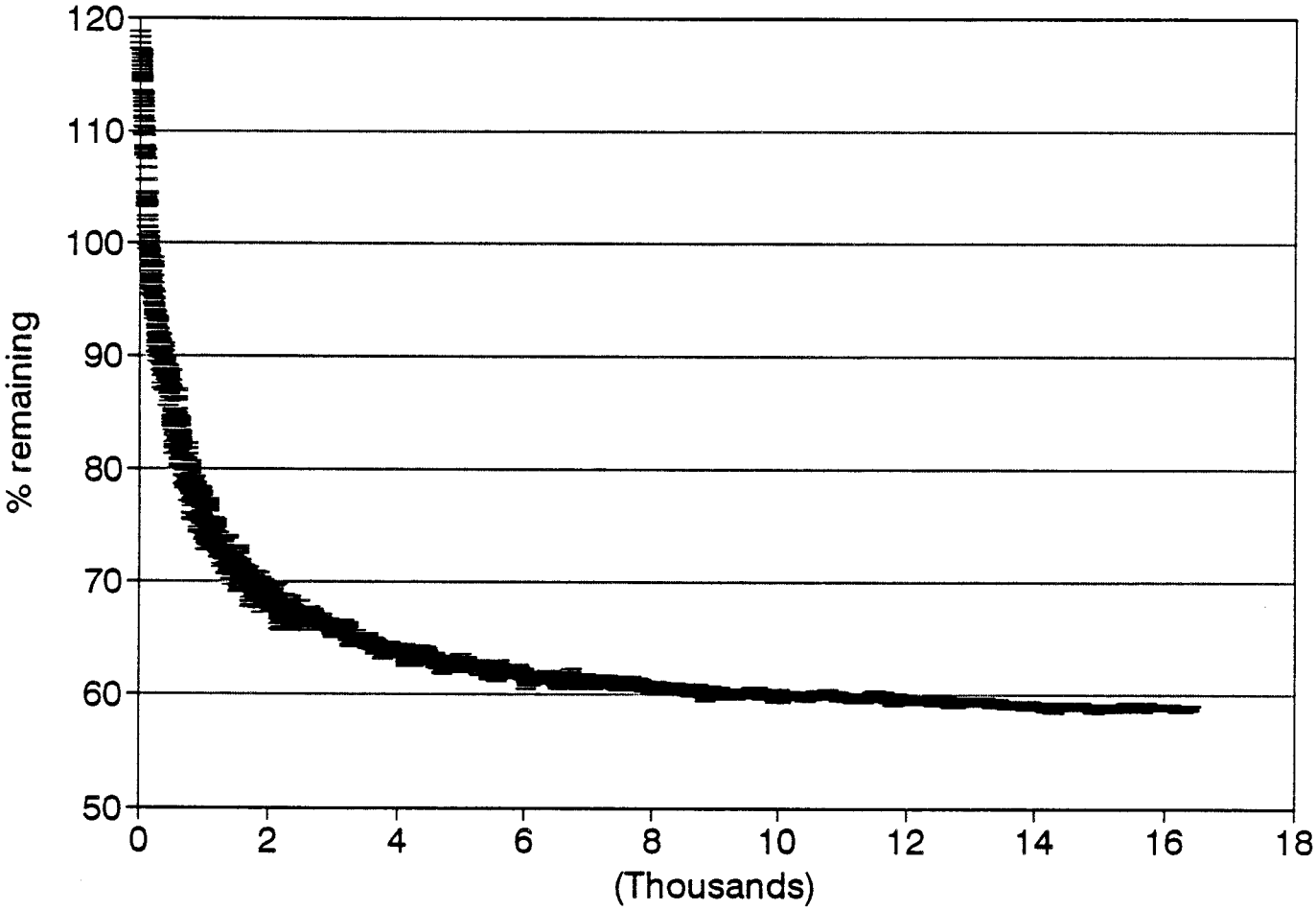

Fig. 2. Individual Records Compression
(Dissertations Abstracts)(LZW1c)