ADAPTIVE TEXT COMPRESSION TECHNIQUE

N.SAIRAM, R.VARADHARAJAN
SASTRA Deemed University,
Thanjavur, India – 613 402.

CHITRA MANIKANDAN
THIAGARAJAR School of Management
Madurai, India – 625 005.

ABSTRACT

In this paper, we propose a simple and efficient lossless compression technique for reducing the size of the text. For typical text files, we achieve about 50% reduction of space. This technique can be used where space is the predominant concern. The intended applications of this technique are files that are frequently transmitted over the net, such as address books, stock quotes, weather forecasting etc. Such files are typically not compressed, but, at the same time, with this scheme they can remain compressed indefinitely, saving space while allowing faster transmission over the net. We also analyze the performance of our technique with other standard text compression techniques.

KEY WORDS: Lossless Compression, Reduction of Space, Saving Space.

1. INTRODUCTION

Text compression is typically used to save storage or communication costs. By reducing the size of a text file in a special way, we can reduce the time of transmission over the net. Our scheme improves the speed of character pair matching and we also save space in this process. The savings are spectacular. The compression consists of substituting each of the common pairs with the special byte allocated for it, and decompression is achieved by reversing this procedure. This kind of compression is as good as adaptive compression techniques (such as the Lempel-Ziv based algorithms [7] or context modeling algorithms [2]).

Files that are usually not compressed, because they are often read, can now be compressed without loss of data and at the same time the speed of transmission can be improved. The improvement is independent of current technology, because it comes from the fact that the compressed files are smaller than the original and therefore less work is done. The same improvement will hold for faster CPU or I/O. Another important advantage of our scheme is that it is independent of the actual character pair -matching program.

Our scheme can be used to compress indexed text using two-level approach. We will assume, throughout the paper, that the search is sequential. Sequential search [6] occurs in many other applications, including DNA and protein search (although in that case approximate matching is used most of the time), bibliographic search, etc. Sequential search also plays a key role in an information-retrieval system[3], based on a two-level approach. For typical text files, we achieve about 50% reduction of space. This technique can be used where space is the predominant concern. The intended applications of this technique are files that are frequently transmitted over the net, such as catalogs, bibliographic files, and address books. Such files are typically not compressed, but with this scheme they can remain compressed indefinitely, at the same time saving space while allowing faster transmission.

Our paper is organized as follows.

In section 2.1, we have proposed a lossless text shrinking algorithm called **Pair-Length Shrinking Algorithm.** In section 2.2, the lossless, shrunk text expansion algorithm called **Pair-Length Expansion Algorithm** is proposed. In Section 3, the algorithms are weighed against other standard text compression schemes. We finally conclude with a discussion of possible applications.

2. PAIR-LENGTH ALGORITHM

2.1 PAIR-LENGTH SHRINKING ALGORITHM

This algorithm is used to create a compressed file. In the first level of the algorithm, A source file is transformed into a file which is slightly compressed using the *code table.* For example, the content of the file AAAABBCBBAA is replaced by ||}C}| with the help of the following code table.

Character	**Code**
A	|
B	}

Table 1. Code Table

In the next level of the algorithm, the slightly compressed file is again compressed by counting the consecutive identical characters and replacing them with their count and character. Now the content of the compressed file becomes 2|}C}|. The diagrammatic representation of the Pair-Length Shrinking Algorithm is depicted in Fig 1.

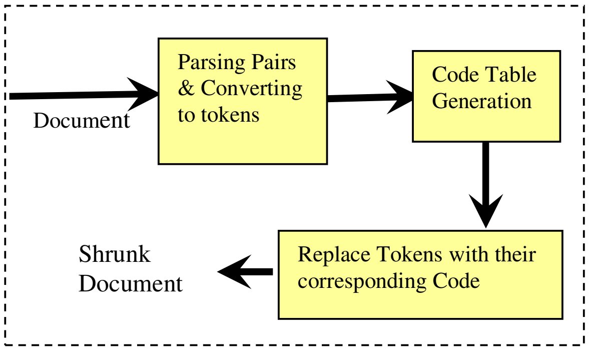
Document
Parsing Pairs & Converting to tokens
Code Table Generation
Replace Tokens with their corresponding Code
Shrunk Document

Figure 1. Pair-Length Shrinking.

```
Algorithm Pair-LenghtShrink(File1)
begin
        Loop until all the characters in File1 are
        exhausted
        begin
           If two consecutive characters are identical
           then
           begin
            Assign a unique code to that character.
            Store that character and the corresponding
            code in the code table.
            Replace the pair by the corresponding
            code in File2.
           end
          else
              Write the character in File2.
        end loop
        Loop until all the characters in File2 are
        exhausted
        begin
                Count the number of consecutive
                identical characters
                if (Count > 1) then
                  Write the Count followed by
                  the corresponding character in
                  File3.
                else
                  Write the character in File3.
        end loop
end.
```

2.2 PAIR-LENGTH EXPANSION ALGORITHM

This algorithm is used to transform a *compressed file* into *its original file*. In the first level of the algorithm, the compressed file is transformed into a file which is slightly expanded by replacing the characters, which are preceded by a number 'n', n times. For example, the content of the compressed file 2|}C}| is replaced by ||}C}| .

In the next level of the algorithm, the slightly expanded file is again expanded by identifying the corresponding character from the code table and replacing the code by the character twice. Now the content of the original file becomes AAAABBCBBAA. The diagrammatic representation of the Pair-Length Expansion Algorithm is depicted in Fig 2.

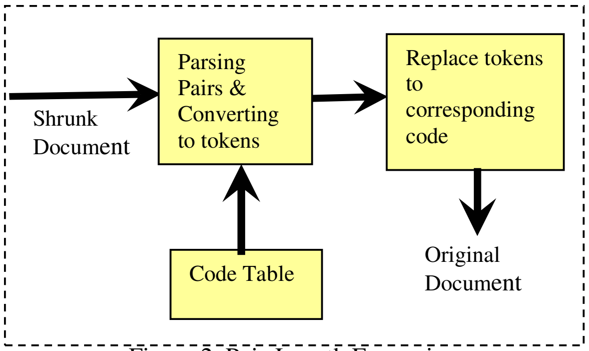
Shrunk Document
Parsing Pairs & Converting to tokens
Replace tokens to corresponding code
Code Table
Original Document

Figure 2. Pair-Length Expansion.

```
Algorithm Pair-LenghtExpand(File1, CT)
begin
        Loop until EOF(File1)
        begin
           If the character ch is numeric then
           begin
                n <-- ch
                Read the next character ch
                Write ch 'n' times in File2
                Search the corresponding character x
                of ch in CT
                Replace each ch by x twice File2.
           end
          else
           begin
                Search the corresponding character x
                of ch in CT
                Replace ch by x twice in File2.
            end
        end loop
end.
```

Where
File1 – Shrunk File
CT – Code Table

2.3 IMPLEMENTATION

We used texts of varying sizes (these texts were also used in Table 2), and also experiments were done with random text. Our program uses a two level approach and as a result the file size is significantly reduced. We selected 100 random words from a dictionary (the effectiveness of this Pair-Length algorithm depends somewhat on the pattern), and *the size of the compressed file, when compared to original file size, is never increased.* We have shown the compression size and compressed rate of various texts in Table 2 (See Appendix).

3. PAIR-LENGTH VS OTHER STANDARD TECHNIQUES

The following table shows the performance of the Pair-Length algorithm when compared with LZW [5, 7] and RLE [2]. Another interesting experiment was done with random text. The compression rates of this algorithm for random text of uniform distribution are quite predictable. In the charts(Fig 3 – 5, see Appendix), the Original size and compressed size of various texts that are listed in Table 2, using RLE, Pair-Length and LZW are depicted.

4. CONCLUSION

In this paper, an efficient compression algorithm is proposed. The compression algorithm is a two-stage algorithm; the first stage reads the file and identifies the consecutive identical pairs. The second stage reads the file again and performs the compression. When our algorithm is compared with other standard compression algorithms like LZW, RLE etc., the size of the compressed file is significantly less. This algorithm can be effectively used to transmit data through mobile network and in all areas where file size is the predominant concern.

REFERENCES

[1]. Khalid Sayood, *Introduction to Data Compression* (San Diego, USA, Academic Press, 1996).

[2]. Bell, T. G., J. G. Cleary, and I. H. Witten, *Text Compression (*Englewood Cliffs, NJ, Prentice- Hall, 1990).

[3]. Manber, U., and S. Wu, ''A two-level approach to information retrieval.'' *Technical report, 93-06*, Department of Computer Science, University of Arizona (March 1993).

[4]. Witten, I. H., T. C. Bell, and C. G. Nevill, ''Models for compression in full-text retrieval systems,'' *Data Compression Conference,* Snowbird, Utah (April 1991), pp. 23−32.

[5]. Welch, T. A., ''A technique for high-performance data compression,'' *IEEE Computer,* 17 (June 1984), pp. 8−19.

[6]. Wu S., and U. Manber, ''Fast Text Searching Allowing Errors,'' *Communications of the ACM* 35 (October 1992), pp. 83−91.

[7]. Ziv, J. and A. Lempel, ''A universal algorithm for sequential data compression,'' *IEEE Trans. on Information Theory,* IT-23 (May 1977). pp. 337−343.

[8] David Salomon, *Data Compression: The Complete Reference* (Springer Verlag, 1998).

[9] Mark Nelson, *The Data Compression Book (* M&T Books, 1995).

[10] James A.Storer, *Image and Text Compression* (Kluwer, 1992).

[11] Mayne, A. and James E.B., "*Information Compression by Factorizing Common Strings*" *The Computer Journal,* Vol. 18, No.2 1974, PP 157 – 160.

APPENDIX

Technique	Text	Original File Size (in Bytes)	Compressed File Size (in Bytes)	Compression Rate
LZW	**AAAABBCBBAA**	**11**	**9.79**	**11%**
	EEEEEEEECCGG	**12**	**8.76**	**27%**
	ZXCWWBCCRR	**10**	**11.4**	**-14%**
	TYTTTTNMUU	**10**	**10.1**	**-1%**
	25667788889	**11**	**11.22**	**-2%**
RLE	**AAAABBCBBAA**	**11**	**13**	**-18%**
	EEEEEEEECCGG	**12**	**9**	**25%**
	ZXCWWBCCRR	**10**	**13**	**-30%**
	TYTTTTNMUU	**10**	**10**	**0%**
	25667788889	**11**	**12**	**-9%**
Pair-Length	**AAAABBCBBAA**	**11**	**6**	**45%**
	EEEEEEEECCGG	**12**	**4**	**66%**
	ZXCWWBCCRR	**10**	**7**	**30%**
	TYTTTTNMUU	**10**	**7**	**30%**
	25667788889	**11**	**7**	**36%**

Table 2: Statistics for the compression and Decompression algorithms

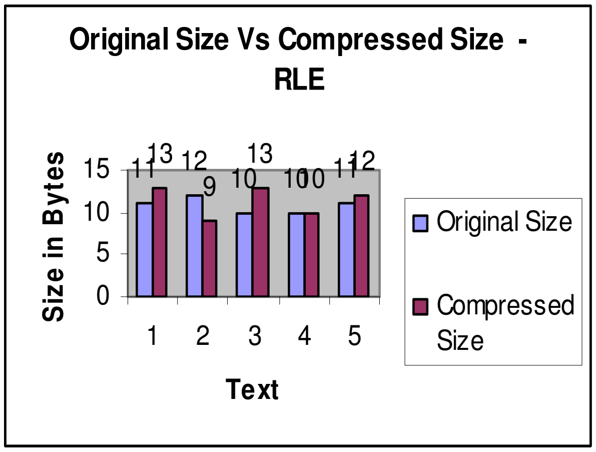

Figure 3. RLE Comparison

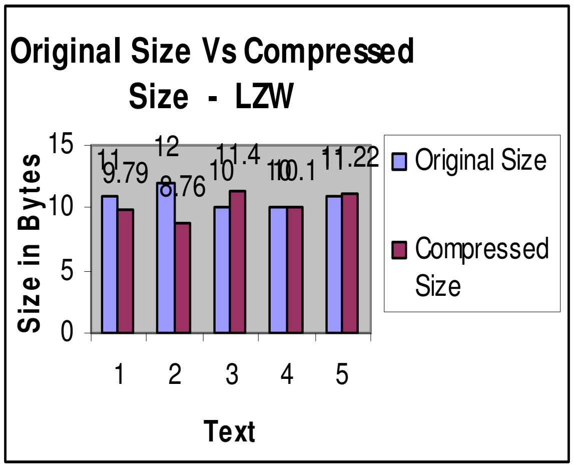

Figure 4. LZW Comparison

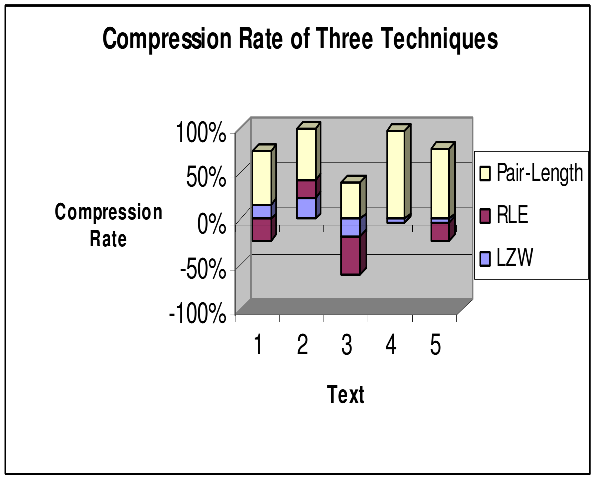

Figure 6. Comparison of Three Techniques

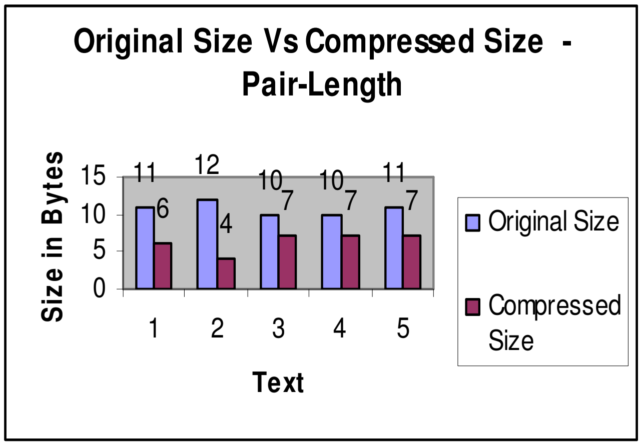

Figure 5. Pair-Length Comparison