ISSN: 2278 – 7798
International Journal of Science, Engineering and Technology Research (IJSETR)
Volume 2, Issue 2, February 2013

A SURVEY ON LOSSLESS DICTIONARY BASED DATA COMPRESSION ALGORITHMS

NISHAD PM, Dr. R.Manicka Chezian

Abstract- Data compression is a common requirement for most of the computerized applications. There are number of data compression algorithms, which are dedicated to compress different data formats. Even for a single data type there are number of different compression algorithms, which use different approaches. This paper presents survey on several dictionary based lossless data compression algorithms and compares their performance based on compression ratio and time ratio on Encoding and decoding. A set of selected algorithms are examined and implemented to evaluate the performance in compressing benchmark text files. An experimental comparison of a number of different dictionary based lossless data compression algorithms is presented in this paper. This paper concluded by stating which algorithm performs well for text data.

Keywords: *Data compression, Encryption, Decryption, Lossless Compression, Lossy Compression*

1. Introduction

Compression is the art of representing the information in a compact form rather than its original or uncompressed form [1]. In other words, using the data compression, the size of a particular file can be reduced. This is very useful when processing, storing or transferring a huge file, which needs lots of resources. If the algorithms used to encrypt works properly, there should be a significant difference between the original file and the compressed file. When data compression is used in a data transmission application, speed is the primary goal. Speed of transmission depends upon the number of bits sent, the time required for the encoder to generate the coded message and the time required for the decoder to recover the original ensemble. In a data storage application, the degree of compression is the primary concern. Compression can be classified as either lossy or lossless. Lossless compression techniques reconstruct the original data from the compressed file without any loss of data. Thus the information does not change during the compression and decompression processes. These kinds of compression algorithms are called reversible compressions since the original message is reconstructed by the decompression process. Lossless compression techniques are used to compress medical images, text and images preserved for legal reasons, computer executable file and so on [2]. Lossy compression techniques reconstruct the original message with loss of some information. It is not possible to reconstruct the original message using the decoding process, and is called irreversible compression [3]. The decompression process results an approximate reconstruction. It may be desirable, when data of some ranges which could not recognized by the human brain can be neglected. Such techniques could be used for multimedia images, video and audio to achieve more compact data compression.

Various dictionary based lossless data compression algorithms have been proposed and used. Some of the main techniques in use are the LZ77, LZR, LZSS, LZH and LZW Encoding and decoding. This paper examines the performance of the above mentioned algorithms are used. In particular, performance of these algorithms in compressing text data is evaluated and compared.

2. Methods and Materials

In order to evaluate the effectiveness and efficiency of dictionary based lossless data compression algorithms the following materials and methods are used.

3 LZ77

Jacob Ziv and Abraham Lempel had introduced a simple and efficient compression method published in their article "A Universal Algorithm for Sequential Data Compression". This algorithm is referred to as LZ77 in honor to the authors and the publishing date 1977. LZ77 is a dictionary based algorithm that addresses byte sequences from former contents instead of the original data. In general only one coding scheme exists; all data will be coded in the same form:

- Address to already coded contents
- Sequence length
- First deviating symbol

Figure -1 sliding window of LZ77

If no identical byte sequence is available from former contents, the address 0, the sequence length 0 and the new symbol will be coded.

Pseudo code Encoding –algorithm

```
while look-ahead buffer is not empty
go backwards in search buffer to find longest match
of the look-ahead buffer
if match found
print: (offset from window boundary, length of match,
next symbol in lookahead
buffer);
shift window by length+1;
else
print: (0, 0, first symbol in look-ahead buffer);
shift window by 1;
fi
end while
```

Pseudo code Decoding –algorithm

```
for each token (offset, length, symbol)
if offset = 0 then
print symbol;
else
go reverse in previous output by offset characters and
copy
character wise for length symbols;
print symbol;
fi
Next
```

Improvements

3.1 LZR

The LZR modification allows pointers to reference anything that has already been encoded without being limited by the length of the search buffer (window size exceeds size of expected input). Since the position and length values can be arbitrarily large, a variable-length representation is being used positions and lengths of the matches.

3.2 LZSS

The mandatory inclusion of the next non-matching symbol into each codeword will lead to situations in which the symbol is being explicitly coded despite the possibility of it being part of the next match. Example: In "abbca|caabb", the first match is a reference to "ca" (with the first non-matching symbol being "a") and the next match then is "bb" while it could have been "abb" if there were no requirement to explicitly code the first non-matching symbol. The popular modification by Storer and Szymanski (1982) removes this requirement. Their algorithm uses fixed-length codeword's consisting of offset (into the search buffer) and length (of the match) to denote references. Only symbols for which no match can be found or where the references would take up more space than the codes for the symbols are still explicitly coded.

Pseudo code LZSS Encoding –algorithm

```
While (lookAheadBuffer not empty)
{
Get a pointer (position, match) to the longest match;
If (length > minimum_mach_length){
Output (pointer_flag, position, length);
Shift the window length characters along;
} else {
Output (SYMBOL_FLAG, first symbol of look
ahead buffer);
Shift the window 1 character along;
}
}
```

3.3 LZB

LZB uses an elaborate scheme for encoding the references and lengths with varying sizes.

3.4 LZH

The LZH implementation employs Huffman coding to compress the pointers.

4 LZ78

The LZ78 is a dictionary-based compression algorithm that maintains an explicit dictionary. The code words output by the algorithm consist of two elements: an index referring to the longest matching dictionary entry and the first non-matching symbol.

In addition to outputting the codeword for storage/transmission, the algorithm also adds the index and symbol pair to the dictionary. When a symbol that not yet in the dictionary is encountered, the codeword has the index value 0 and it is added to the dictionary as well. With this method, the algorithm gradually builds up a dictionary.

```
w := NIL;
while (there is input){
K := next symbol from input;
if (wK exists in the dictionary) {
w := wK;
} else {
output (index(w), K);
add wK to the dictionary;
```

```
w := NIL;
}
}
```

Note that this simplified pseudo-code version of the algorithm does not prevent the dictionary from growing forever. There are various solutions to limit dictionary size, the easiest being to stop adding entries and continue like a static dictionary coder or to throw the dictionary away and start from scratch after a certain number of entries has been reached.

4.1 LZW

The LZW algorithm uses dictionary while decoding and encoding but the time taken for creating the dictionary is large, so to reduce the time complexity a new methodology is proposed in this paper. The number of shift before of a new pattern and the number of comparison required to find the pattern in the dictionary is reduced after the implementation of multiple dictionaries. The experimental result shows massive reduction in the time complexity.

LZW compression uses a code table common choice is to provide 4096 entries in the table. In this case, the LZW encoded data consists of 12 bit codes, each referring to one of the entries in the code table. Decompression is achieved by taking each code from the compressed file, and translating it through the code table to find what character or characters it represents. Codes 0-255 in the code table are always assigned to represent single byte from the input file. When the LZW program starts to encode a file, the code table contains only the first 256 entries, with the remainder of the table being blank. This means that the first code in the compressed file is of single byte from the input file being converted to 12 bits. As the encoding continues, the LZW algorithm identifies repeated sequences in the data, and adds them to the code table. Compression starts the second time a sequence is encountered. The key point is that a sequence from the input file is not added to the code table until it has already been placed in the compressed file as individual characters (codes 0 to 255). This is important because it allows the decompression program to reconstruct the code table directly from the compressed data, without having to transmit the code table separately.

The compression algorithm uses two variables: *CHAR* and *STRING*. The variable, *CHAR*, holds a single character, (i.e.), a single byte value between 0 and 255. The variable, *STRING*, is a variable length string, (i.e.), a group of one or more characters, with each character being a single byte. The algorithm starts by taking the first byte from the input file, and placing it in the variable, *STRING*. Table -1 show this action in line 1. This is followed by the algorithm looping for each additional byte in the input file. Each time a byte is read from the input file it is stored in the variable, *CHAR*. The data table is then searched to determine if the concatenation of the two variables, *STRING+CHAR*, has already been assigned a code. If a match in the code table is not found, three actions are taken, (i), output the code for STRING, When a match in the code table is found, (ii), the concatenation of *STRING+CHAR* is stored in the variable, *STRING*, without any other action taking place. That is, if a matching sequence is found in the table, no action should be taken before determining whether there is a longer matching sequence is present in the table or not. An example of this is shown in line 5, where the sequence: *STRING+CHAR* = *'AB'*, is identified as already having a code in the table. In line 6, the next character from the input file, *'B'*, is added to the sequence, and the code table is searched for: *'ABB'*. Since this longer sequence is not in the table, the algorithm adds it to the table, outputs the code for the shorter sequence that is in the table (code 256), and starts over searching for sequences beginning with the character, *'B'*. This flow of events is continued until there are no more characters in the input file. The program is wrapped up with the code corresponding to the current value of *STRING* being written to the compressed file. LWZ compression algorithm is illustrated in Table-1. The Decompression algorithm uses four variables NCODE, OCODE, STRING, and CHAR. The decompression algorithm starts by taking the first byte from the input file and placing it in the variable, OCODE and output the OCODE. This action is shown in table-2 line 1. This is followed by the algorithm looping for each additional byte in the input file; each time a byte is read from the input file it is stored in the variable, NCODE. The data table is then searched to find the variable NCODE. If a match in the code table is not found STRING = OCODE +CHAR else if the NCODE is found then STRING = NCODE, then output the STRING. First Character of STRING is assigned to CHAR, then adds entry (OCODE+ CHAR) in table for and assigns NCODE to OCODE. This process will continue up to the last input. The decoding algorithm is shown in table-2.

TABLE -1

LZW example: This shows the compression of the phrase: *ABCABBABBBCABABBBD*

S no	Char	String +Char	In Table?	Out put	Add to Table	New String	Commends
1	A	A				A	first character- no action
2	B	AB	No	A	256 =AB	B	
3	C	BC	No	B	257 =BC	C	
4	A	CA	No	C	258 =CA	A	
5	B	AB	Yes(256)			AB	first match found
6	B	ABB	No	256	259=ABB	B	
7	A	BA	No	B	260 =BA	A	
8	B	AB	Yes(256)			AB	Matches
9	B	ABB	Yes (259)			ABB	Longest Match
10	B	ABBB	No	259	261=ABBB	B	Longest Match index Substituted
11	C	BC	Yes(257)			BC	Matches
12	A	BCA	No	257	262=BCA	A	
13	B	AB	Yes(256)			AB	Matches
14	A	ABA	No	256	263=ABA	A	
15	B	AB	Yes(256)			AB	Matches
16	B	ABB	Yes(259)			ABB	Matches
17	B	ABBB	Yes(261)			ABBB	Longest Match
18	D	ABBBD	No	261	264=ABBBD	D	Longest Match index Substituted
19	EOF	D	No	D			End of file, Output string.

The Existing implimentation of LZW, with BST and simple binary search has several limitations that directly leads to the time complexity. in LZW the comparison ratio riquired for the new pattern while encoding (NCODE) and decoding(OLDCODE+CHAR) is huge. For example if NCODE and OLDCODE+CHAR is ‘ABBBD’ then the absence of the pattern is returended after comparing all the elements in the dictionary (shown in table 1 line number -16 for encoding and table-2 line number 10 for decoding). in binary search tree Implimentation (BST) of LZW the search for the pattern ‘ABBBD’ the comparison required through ‘AB’,’BC’,’ABB’,’BA’, ‘ABBB’ then only the additional node is updated in the tree shown in figure-2. in simple binary seearch the Comparison ratio and Shifting before the insertion in huge shown that directly leads to tme complexity.

TABLE -2 LZW examples: This shows the decompression of the phrase: *A B C 256 B 259 257 256 261 D*

S No	NCODE	OCODE	CHAR	STRING/ Output	New table entry
1	A	A	A	A	
2	B	A	B	B	256=AB
3	C	B	C	C	257=BC
4	256	C	A	AB	258=CA
5	B	256	B	B	259=ABB
6	259	B	A	ABB	260=BA
7	257	259	B	BC	261=ABBB
8	256	257	A	AB	262=BCA
9	261	256	A	ABBB	263=ABA
10	D	261	D	D	264= ABBBD

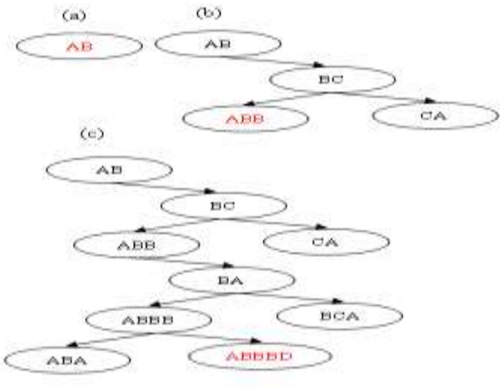

Figure 2 BST insertion for LZW encoding and decoding (a) After insertion of 'AB'. (b) After insertion of 'ABB'. (c) After insertion of 'ABBBD'.

Pseudo code for LZW Encoding –algorithm

```
STRING = get input CHAR
WHILE there are still input CHAR DO
    CHARACTER = get input CHAR
    IF STRING+CHAR is in the string table then
        STRING = STRING+CHAR
    ELSE
        Output the code for STRING
        Add STRING+CHAR to the string table
        STRING = CHAR
    END of IF
END of WHILE
output the code for STRING
```

Pseudo code for LZW Decoding–algorithm

```
Read OCODE
  output OCODE
    WHILE there are still input characters DO
```

```
        Read NCODE
        STRING = get translation of NCODE
        output STRING
        CHAR = first character in STRING
        add OLD_CODE + CHAR to the translation table
        OLD_CODE = NEW_CODE
    END of WHILE
```

The GIF Controversy GIF image compression is probably the first thing that comes to mind for most people who have ever heard about the Lempel Ziv algorithm. Originally developed by CompuServe in the late 1980s, the GIF file format employs the LZW technique for compression. Apparently, CompuServe designed the GIF format without knowing that the LZW algorithm was patented by Unisys.

For years, the GIF format, as well as other software using the LZW algorithm, existed peacefully and became popular, without ever being subject to licensing fees by the patent holder. Then in 1994, Unisys announced a licensing agreement with CompuServe, and subsequently started demanding royalties from all commercial software developers selling software that incorporated LZW compression. Later the royalty demand was extended to non-commercial software, sparking an even greater outrage among the internet and software development community than the initial announcement. Demands to "Burn all GIFs" and efforts to produce a patent-free alternative to GIF, PNG, received considerable attention, but nevertheless GIF continues to be popular. The patent on the LZW algorithm will expire in June 2003. Still, several other algorithms of the Lempel Ziv family remain protected by patents. Jean-loup Gailly, the author of the gzip compression program has done extensive research into compression patents

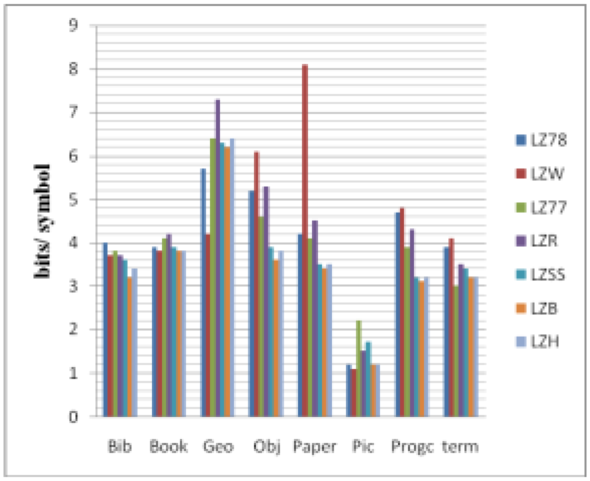

Chart 1: Comparison LZ77 and LZ78

5. Comparison

The following chart shows a comparison of the compression rates for the Different LZ77 and LZ78 variants. The compression rate is measured in bits/symbol, indicating how many bits are needed on average to encode a symbol (for binary files: symbol = byte).

6. Conclusions

An experimental comparison of a number of different dictionary based lossless compression algorithms for text data is carried out. Several existing lossless compression methods are compared for their effectiveness. Although they are tested on different type of files, the main interest is on different test patterns. By considering the compression ratio, the LZW algorithm may be considered as the most efficient algorithm among the selected ones. Those values of this algorithm are in an acceptable range and it shows better results.

References

[1]Pu, I.M., 2006, *Fundamental Data Compression*, Elsevier, Britain.

[2] Blelloch, E., 2002. *Introduction to Data Compression*, Computer Science Department, Carnegie Mellon University.

[3] Kesheng, W., J. Otoo and S. Arie, 2006. *Optimizing bitmap indices with efficient compression*, ACM Trans. Database Systems, 31: 1-38.

[4] Kaufman, K. and T. Shmuel, 2005. *Semi-lossless text compression*, Intl. J. Foundations of Computer Sci., 16: 1167-1178.

[5] Campos, A.S.E, *Basic arithmetic coding by Arturo Campos Website*, Available from: http://www.arturocampos.com/ac_arithmetic.html. (Accessed 02 February 2009)

[6] Vo Ngoc and M. Alistair, 2006. *Improved wordaligned binary compression for text indexing*, IEEE Trans. Knowledge & Data Engineering, 18: 857-861.

[7] Cormak, V. and S. Horspool, 1987. *Data compression using dynamic Markov modeling*, Comput. J., 30: 541–550.

[8] Capocelli, M., R. Giancarlo and J. Taneja, 1986. *Bounds on the redundancy of Huffman codes*, IEEE Trans. Inf. Theory,32: 854–857.

[9] Gawthrop, J. and W. Liuping, 2005. *Data compression for estimation of the physical parameters of stable and unstable linear systems*, Automatica, 41: 1313-1321.

[10] comp . compression FAQ. http: // www. faqs.org/faqs/compressionfaq/.

[11] Bell,T.C.,Clearly, J. G., AND Witten, I. H. Text Compression. Prentice Hall, Upper Sadle River, NJ, 1990.

[12] Sayood, K. “Introduction to Data Compression”. Academic Press, San Diego, CA, 1996, 2000.
[13] Ziv, J., and Lempel, A. “A universal algorithm for sequential data Compression”. IEEE Transactions on Information Theory 23 (1977), 337–343.
[14] ZIV, J., AND LEMPEL, A. Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory 24 (1978), 530–536.

Biography

First Author Nishad PM M.Sc., M.Phil. Seven months Worked as a project trainee in Wipro in 2005, five years experience in teaching, one and half year in JNASC and Three and half year in MES Mampad College. He has published eight papers national level/international conference and journals. He has presented three seminars at national Level. Now he is pursuing Ph.D Computer Science in Dr. Mahalingam center for research and development at NGM College Pollachi.

Second Author Dr. R.Manicka chezian received his M.Sc., degree in Applied Science from P.S.G College of Technology, Coimbatore, India in 1987. He completed his M.S. degree in Software Systems from Birla Institute of Technology and Science, Pilani, Rajasthan, India and Ph D degree in Computer Science from School of Computer Science and Engineering, Bharathiar University, Coimbatore, India. He served as a Faculty of Maths and Computer Applications at P.S.G College of Technology, Coimbatore from 1987 to 1989. Presently, he has been working as an Associate Professor of Computer Science in N G M College (Autonomous), Pollachi under Bharathiar University, Coimbatore, India since 1989. He has published thirty papers in international/national journal and conferences: He is a recipient of many awards like Desha Mithra Award and Best Paper Award. His research focuses on Network Databases, Data Mining, Distributed Computing, Data Compression, Mobile Computing, Real Time Systems and Bio-Informatics.