Comparative Study of Dictionary based Compression Algorithms on Text Data

Amit Jain **Kamaljit I. Lakhtaria**

Sir Padampat Singhania University, Udaipur (Raj.) 323601 India

Abstract:
With increasing amount of text data being stored rapidly, efficient information retrieval and Storage in the compressed domain has become a major concern. Compression is the process of coding that will effectively reduce the total number of bits needed to represent certain information. Data compression has been one of the critical enabling technologies for the ongoing digital multimedia revolution. There are lots of data compression algorithms which are available to compress files of different formats. 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. The paper is concluded by the decision showing which algorithm performs best over text data.

Keywords -
LZW, Dictionary Encoding, Compression Ratio, Compression time.

1. Introduction

Compression is the art of representing information in a compact form rather than its original or uncompressed form [1]. The main objective of data compression is to find out the redundancy and eliminate them through different efficient methodology; so that the reduced data can save, space: to store the data, time: to transmit the data and cost: to maintain the data. To eliminate the redundancy, the original file is represented with some coded notation and this coded file is known as ‘encoded file’. For any efficient compression algorithm this file size must be less than the original file. To get back the original file we need to ‘decode’ the encoded file

Types of Compression:
Compression can be of two types: Lossless Compression, Lossy Compression.

Lossless Compression:
In the process compression if no data is lost and the exact replica of the original file can be retrieved by decrypting the encrypted file then the compression is of lossless compression type. Text compression is generally of lossless type. In this type of compression generally the encrypted file is used for storing or transmitting data, for general purpose use we need to decrypt the file. Lossless compression technique can be broadly categorized in to two classes:

i) Entropy Based Encoding:
In this compression process the algorithm first counts the frequency of occurrence of each unique symbol in the document. Then the compression technique replaces the symbols with the algorithm generated symbol. These generated symbols are fixed for a certain symbol of the original document; and doesn’t depend on the content of the document. The length of the generated symbols is variable and it varies on the frequency of the certain symbol in the original document.

ii) Dictionary Based Encoding:
This encoding process is also known as substitution encoding. In this process the encoder maintain a data structure known as ‘Dictionary’ [2]. This is basically a collection of string. The encoder matches the substrings chosen from the original text and finds it in the dictionary; if a successful match is found then the substring is replaced by a reference to the dictionary in the encoded file.
There are quite a few lossless compression techniques nowadays, and most of them are based on dictionary or probability and entropy. In other words, they all try to utilize the occurrence of the same character/string in the data to achieve 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.
The paper is organized as follows: Section I contains a brief Introduction about compression and its types, Section II presents a brief explanation about different dictionary based compression techniques, Section III has its focus on comparing the performance of compression techniques and the final section contains the Conclusion.

2. Data Compression Techniques

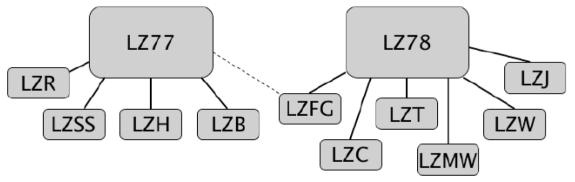
Various kind of text data compression algorithms have been proposed till date, mainly those algorithms is lossless algorithm. Dictionary coding techniques rely upon the observation that there are correlations between parts of data (recurring patterns). The basic idea is to replace those repetitions by (shorter) references to a "dictionary" containing the original. This paper examines the performance of the while family of of Lempel Ziv

Manuscript received February 5, 2016
Manuscript revised February 20, 2016

Algorithm. Performance of above listed algorithms for compressing text data is evaluated and compared.

2.1 LEMPEL ZIV ALGORITHMS

The Lempel Ziv Algorithm is an algorithm for lossless data compression. It is not a single algorithm, but a whole family of algorithms, stemming from the two algorithms proposed by Jacob Ziv and Abraham Lempel in their landmark papers in 1977 and 1978.



2.1.1 LZ77:

Jacob Ziv and Abraham Lempel have presented their dictionary-based scheme in 1977 for lossless data compression [3]. Today this technique is much remembered by the name of the authors and the year of implementation of the same.

LZ77 exploits the fact that words and phrases within a text file are likely to be repeated. When there is repetition, they can be encoded as a pointer to an earlier occurrence, with the pointer accompanied by the number of characters to be matched. It is a very simple adaptive scheme that requires no prior knowledge of the source and seems to require no assumptions about the characteristics of the source.

In the LZ77 approach, the dictionary is simply a portion of the previously encoded sequence. The encoder examines the input sequence through a sliding window which consists of two parts: a search buffer that contains a portion of the recently encoded sequence and a look ahead buffer that contains the next portion of the sequence to be encoded. The algorithm searches the sliding window for the longest match with the beginning of the look-ahead buffer and outputs a reference (a pointer) to that match. It is possible that there is no match at all, so the output cannot contain just pointers. In LZ77 the reference is always output as a triple <o,l,c>, where ‘o’ is an offset to the match, ‘l’ is length of the match, and ‘c’ is the next symbol after the match. If there is no match, the algorithm outputs a null-pointer (both the offset and the match length equal to 0) and the first symbol in the look-ahead buffer[4].

The values of an offset to a match and length must be limited to some maximum constants. Moreover the compression performance of LZ77 mainly depends on these values. Usually the offset is encoded on 12–16 bits, so it is limited from 0 to 65535 symbols. So, there is no need to remember more than 65535 last seen symbols in the sliding window. The match length is usually encoded on 8 bits, which gives maximum match length equal to 255[5].

The LZ77 algorithm is given below:

```
While (lookAheadBuffer not empty) {
get a reference (position, length) to longest match;
if (length > 0) {
output (position, length, next symbol);
shift the window length+1 positions along;
}
else {
output (0, 0, first symbol in the lookahead buffer);
shift the window 1 character along;
}
}
```

With regard to other algorithms the time for compression and decompression is just the same. In LZ77 encoding process one reference (a triple) is transmitted for several input symbols and hence it is very fast. The decoding is much faster than the encoding in this process and it is one of the important features of this process. In LZ77, most of the LZ77 compression time is, however, used in searching for the longest match, whereas the LZ77 algorithm decompression is quick as each reference is simply replaced with the string, which it points to.
There are lots of ways that LZ77 scheme can be made more efficient and many of the improvements deal with the efficient encoding with the triples. There are several variations on LZ77 scheme, the best known are LZSS, LZH and LZB. LZSS which was published by Storer and Szymanksi [6] removes the requirement of mandatory inclusion of the next non-matching symbol into each codeword. Their algorithm uses fixed length codewords consisting of offset and length to denote references. They propose to include an extra bit (a bit flag) at each coding step to indicate whether the output code represents a pair (a pointer and a match length) or a single symbol.
LZH is the scheme that combines the Ziv – Lempel and Huffman techniques. Here coding is performed in two passes. The first is essentially same as LZSS, while the second uses statistics measured in the first to code pointers and explicit characters using Huffman coding.
LZB was published by Mohammad Banikazemi[7] uses an elaborate scheme for encoding the references and lengths with varying sizes. Regardless of the length of the phrase it represents, every LZSS pointer is of the same size. In practice a better compression is achieved by having different sized pointers as some phrase lengths are much more likely to occur than others. LZB is a technique that uses a different coding for both components of the pointer.

LZB achieves a better compression than LZSS and has the added virtue of being less sensitive to the choice of parameters.

2.1.2 LZ78

In 1978 Jacob Ziv and Abraham Lempel presented their dictionary based scheme [8], which is known as LZ78. It is a dictionary based compression algorithm that maintains an explicit dictionary. This dictionary has to be built both at the encoding and decoding side and they must follow the same rules to ensure that they use an identical dictionary. The codewords output by the algorithm consists of two elements <i,c> where 'i' is 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 is not yet found in the dictionary, the codeword has the index value 0 and it is added to the dictionary as well. The algorithm gradually builds up a dictionary with this method.
The algorithm for LZ78 is given below:

```
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;
}
}
```

LZ78 algorithm has the ability to capture patterns and hold them indefinitely but it also has a serious drawback. The dictionary keeps growing forever without bound. There are various methods 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. The encoding done by LZ78 is fast, compared to LZ77, and that is the main advantage of dictionary based compression. The important property of LZ77 that the LZ78 algorithm preserves is the decoding is faster than the encoding. The decompression in LZ78 is faster compared to the process of compression.

2.1.3 LZW

Terry Welch has presented his LZW (Lempel–Ziv–Welch) algorithm in 1984[9], which is based on LZ78. It basically applies the LZSS principle of not explicitly transmitting the next non-matching symbol to LZ78 algorithm. The dictionary has to be initialized with all possible symbols from the input alphabet. It guarantees that a match will always be found. LZW would only send the index to the dictionary. The input to the encoder is accumulated in a pattern 'w' as long as 'w' is contained in the dictionary. If the addition of another letter 'K' results in a pattern 'w*K' that is not in the dictionary, then the index of 'w' is transmitted to the receiver, the pattern 'w*K' is added to the dictionary and another pattern is started with the letter 'K'.

The algorithm then proceeds as follows:

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

In the original proposal of LZW, the pointer size is chosen to be 12 bits, allowing for up to 4096 dictionary entries. Once the limit is reached, the dictionary becomes static.

LZFG which was developed by Fiala and Greene [10], gives fast encoding and decoding and good compression without undue storage requirements. This algorithm uses the original dictionary building technique as LZ78 does but the only difference is that it stores the elements in a trie data structure. Here, the encoded characters are placed in a window (as in LZ77) to remove the oldest phrases from the dictionary.

3. Experimental Results

In this section we focus our attention to compare the performance of LZ77 family algorithms (LZ77, LZSS, LZH and LZB) and LZ78 family algorithms (LZ78, LZW and LZFG). Research works done to evaluate the efficiency of any compression algorithm are carried out having two important parameters. One is the amount of compression achieved and the other is the time used by the encoding and decoding algorithms. We have tested several times the practical performance of the above mentioned techniques on files of Canterbury corpus and have found out the results of Lempel –Ziv techniques selected for this study.

LZ algorithms considered here are divided into two categories: those derived from LZ77 and those derived from LZ78. The BPC measurements are referred from [11]. Table – I shows the comparison of various algorithms that are derived from LZ77(LZ77, LZSS, LZH and LZB). Table – II shows the comparative analysis of algorithms that are derived from LZ78 (LZ78, LZW and LZFG). The BPC values that are referred from [11] are based on the following parameters.

The main parameter for LZ77 family is the size of the window on the text. Compression is best if the window is as big as possible but not bigger than the text, in general. Nevertheless, larger windows yield diminishing returns. A window as small as 8000 characters will perform much better, and give a result nearly as good as the ones derived from the larger windows. Another parameter which limits the number of characters is needed for some algorithms belonging to LZ family. Generally a limit of around 16 may work well. For LZ77, LZSS and LZB the storage (characters in window) were assumed to be of 8 KB and for LZH it was assumed as 16 KB. Regarding LZ78 family, most of the algorithm requires one parameter to denote the maximum number of phrases stored. For the above mentioned LZ78 schemes, except LZ78 a limit of 4096 phrases was used. Fig 1 shows a comparison of the compression rates for the different LZ77 variants.

Table I. Comparison of BPC for the different LZ77 variants

S. No.	File Name	File Size	LZ77 BPC	LZSS BPC	LZH BPC	LZB BPC
1	Bib	111261	3.75	3.35	3.24	3.17
2	Book1	768771	4.57	4.08	3.73	3.86
3	Book2	610856	3.93	3.41	3.34	3.28
4	News	377109	4.37	3.79	3.84	3.55
5	Obj1	21504	5.41	4.57	4.58	4.26
6	Obj2	246814	3.81	3.3	3.19	3.14
7	Paper1	53161	3.94	3.38	3.38	3.22
8	Paper2	82199	4.1	3.58	3.57	3.43
9	Progc	39611	3.84	3.24	3.25	3.08
10	Prog1	71646	2.9	2.37	2.2	2.11
11	Progp	49379	2.93	2.36	2.17	2.08
12	Trans	93695	2.98	2.44	2.12	2.12
Average BPC			3.88	3.32	3.22	3.11

The output of Table – I reveals that the Bits Per Character is significant and most of the files have been compressed to a little less than half of the original size of LZ77 family, the performance of LZB is significant compared to LZ77, LZSS and LZH. The average BPC which is significant as, shown in Table – I, which is 3.11.

Amongst the performance of the LZ77 family, LZB outperforms LZH. This is because, LZH generates an optimal Huffman code for pointers whereas LZB uses a fixed code.

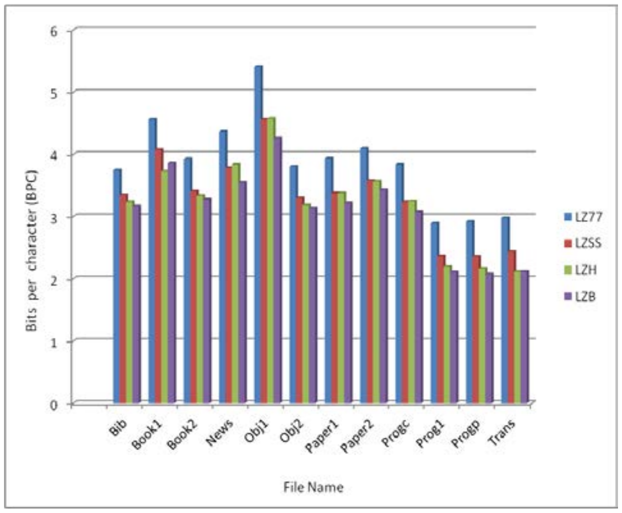


Fig 1 Comparison of the compression rates for the different LZ77 variants

Table II. Comparison of BPC for the different LZ78 variants

S. No.	File Name	File Size	LZ78 BPC	LZW BPC	LZFG BPC
1	Bib	111261	3.95	3.84	2.9
2	Book1	768771	3.92	4.03	3.62
3	Book2	610856	3.81	4.52	3.05
4	News	377109	4.33	4.92	3.44
5	Obj1	21504	5.58	6.3	4.03
6	Obj2	246814	4.68	9.81	2.96
7	Paper1	53161	4.5	4.58	3.03
8	Paper2	82199	4.24	4.02	3.16
9	Progc	39611	4.6	4.88	2.89
10	Prog1	71646	3.77	3.89	1.97
11	Progp	49379	3.84	3.73	1.9
12	Trans	93695	3.92	4.24	1.76
Average BPC			4.26	4.90	2.89

We have tried to infer from Table – II the compression performance of LZ78 family. Most of the
ASCII files are compressed to just less than half of the original size and within each file the amount of compression is consistent. The LZW method, having no boundary, accepts phrases and so the compression expands the file ‘obj2’ by 25%, which is considered as a weakness of this approach. Also from Table – II it is obvious that the performance of LZFG is the best amongst these methods, giving an average BPC of 2.89 which is really significant. Amongst LZ78 family, LZFG’s performance is the best because the scheme that it uses is carefully selected codes to represent a pointer which is like the best scheme in the LZ77 family. Fig 2 represents a comparison of the compression rates for the LZ78 family.

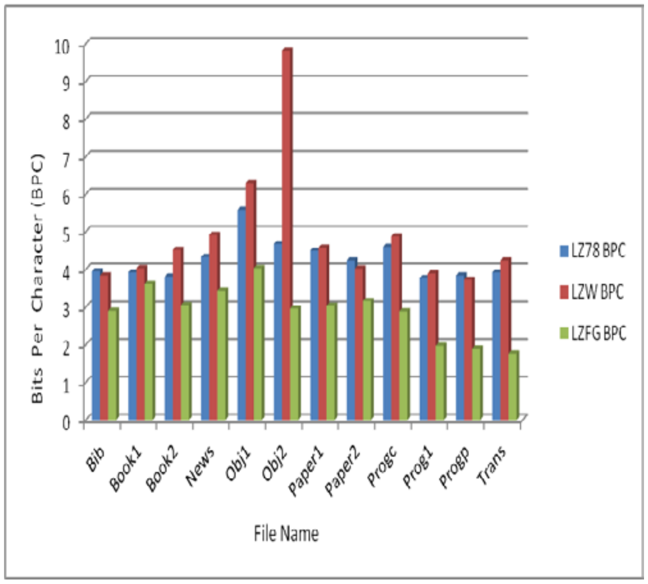


Fig 2 Comparison of the compression rates for the different LZ78 variants

4. Conclusion

We have taken up Lempel Ziv algorithms for our study to examine the performance in compression. In the dictionary based compression techniques, LZB outperforms LZ77, LZSS and LZH to show a marked compression, which is 19.85% improvement over LZ77, 6.33% improvement over LZSS and 3.42% improvement over LZH, amongst the LZ77 family. LZFG shows a significant result in the average BPC compared to LZ78 and LZW. From the result it is evident that LZFG has outperformed the other two with an improvement of 32.16% over LZ78 and 41.02% over LZW.

References

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

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

[3] Ziv. J and Lempel A., “A Universal Algorithm for Sequential Data Compression”, IEEE Transactions on Information Theory 23 (3), pp. 337–342, May 1977.

[4] Khalid Sayood, “Introduction to Data Compression”, 2nd Edition, San Francisco, CA, Morgan Kaufmann, 2000.

[5] Przemyslaw Skibinski, “Reversible Data transforms that improve effectiveness of universal lossless data compression”, Ph.D thesis, Department of Mathematics and Computer Science, University of Wroclaw, 2006.

[6] Storer J and Szymanski T.G., “Data compression via textual substitution”, Journal of the ACM 29, pp. 928–951, 1982.

[7] Mohammad Banikazemi, “LZB: Data Compression with Bounded References”, Proceedings of the 2009 Data Compression Conference, IEEE Computer Society, 2009.

[8] Ziv. J and Lempel A., “Compression of Individual Sequences via Variable-Rate Coding”, IEEE Transactions on Information Theory 24 (5), pp. 530–536, September 1978.

[9] Welch T.A., “A technique for high-performance data compression”, IEEE Computer, 17, pp. 8–19, 1984.

[10] Fiala E.R., and D.H. Greene, “Data Compression with finite windows”, Communications of the ACM 32(4):490-505, 1989.

[11] Bell T.C, Cleary J.G, and Witten I.H., “Text Compression”, Prentice Hall, Upper Saddle River, NJ, 1990.

Amit Jain is working in CSE Department, Sir Padampat Singhania University, Udaipur, India. He is having 17 years of teaching experience. He has taught to post-graduate and graduate students of engineering. He is pursuing Ph.D. in Computer Science, in the area of Information Security. He has presented 3 papers in International Journal, 5 papers in International Conference and 8 papers in National Conference.

Kamaljit I Lakhtaria is working in CSE Department, Sir Padampat Singhaniya University, India. He obtained Ph. D. in Computer Science; area of Research is “Next Generation Networking Service Prototyping & Modeling”. He holds an edge in Next Generation Network, Web Services, MANET, Web 2.0, Distributed Computing. His inquisitiveness has made him present 18 Papers in International Conferences, 28 Paper in International Journals. He is author of 8 Reference Books. He is member of Life time member ISTE, IAENG. He holds the post of Editor, Associate Editor in many International Research Journal. He is reviewer in IEEE WSN, Inderscience and Elsevier Journals.