

A Review of Lempel Ziv Compression Techniques

¹Shalini Kapoor, ²Ashish Chopra

^{1,2}Dept. of CSE, Doon Valley Institute of Engineering & Technology, Karnal, Haryana, India

Abstract

With increasing amount of text data being stored, efficient information retrieval and Storage in the compressed domain has become a major concern. I have studied various Compression algorithms like LZ77 and LZ78. During the study I noticed that the Dictionary based Compression algorithms have several drawbacks.

This Paper direct several key issues to the dictionary based LZW algorithm existing today. In contrast to the Previous LZW, we would like to improve LZW algorithm in future which definitely get good results like: Better compression ratio, time taken for searching in the dictionary for pattern matching in encoding and decoding got reduced and dictionary size become Dynamic. In the future we would like to present modifications possible in existing Lempel Ziv Welch Algorithm by changing its Data structure.

Keywords

LZW, Compression, Dictionary, RLE, Huffman Tree, Compression Ratio, Sliding Window.

I. Introduction

The term "Data Compression" means compresses the data as much as possible from its original size. But in this communication we always want to transmit data efficiently and noise free. Data compression has important applications in the areas of data transmission and data storage despite of the large capacity storage devices that are available these days. Hence, we need an efficient way to store and transmit different types of data such as text, image, audio, and video to reduce execution time and memory size.

The two basic classes of data compression are applied in different areas currently that are lossy and lossless compression. One of the lossless data compression widely used is LZW data compression [5], it is a dictionary based algorithm.

The general principle of data compression algorithms on text files is to transform a string of characters into a new string which contains the same information but with new length as small as possible. The efficient data compression algorithm is chosen according to some scales like: compression size, Compression ratio, processing time or speed [7].

There are quite a few lossless Compression techniques now a days, and most of them are based on dictionary or Probability. In other words, they all try to utilize the occurrence of the same character/string in the data to achieve compression.

In 1980, Terry Welch [1] invented LZW algorithm which became the popular technique for general purpose compression systems. Now-a-days, compression ratio is a great factor in transmission of data. By this research we can have a better solution about how to make compression ratio higher, because data transmission mostly depends on compression ratio.

This paper is organized as follows: Section I contains a brief Introduction about Compression, Section II presents a brief explanation about Compression techniques like Statistical compression techniques, Section III discusses about Dictionary based compression techniques, Section IV focus on LZW Algorithm Compression Process, Section V mainly focus on the previous work done on LZ Algorithms field, Section VI has its focus on the key issues of Lempel Ziv Welch compression

algorithm area and the final section contains the Conclusion and future scope of Improving LZW Algorithm for better Compression performance.

II. Compression Techniques

In this section, we will give a short review and explanation for each one of the Compression techniques that can be used on any text files. Compression algorithms have a long history, the roots of compression algorithms goes back to earlier twentieth century. We however would describe compression systems that are mostly used in currently computation domain for the data Compression:

A. Stastical Compression Techniques

1. Run Length Encoding (RLE)

The Run-Length Encoding (RLE)[5] is created especially for data with strings of repeated symbols (the length of the string is called a run). The main idea behind this is to encode repeated symbols as a pair: the length of the string and the symbol. The biggest problem with RLE is that in the worst case the size of output data can be two times more than the size of input data. To eliminate this problem, each pair can be later encoded with an algorithm like Huffman coding.

B. Huffman Coding

The Huffman coding algorithm [9] is named after its inventor, David Huffman, who developed this algorithm as a student in a class on information theory at MIT in 1950. It is a more successful method used for text

Compression. The Huffman's idea is to replace fixed-length codes (such as ASCII) by variable-length codes, assigning shorter code words to the more frequently occurring symbols and thus decreasing the overall length of the data. When using variable-length code words, it is desirable to create a (uniquely decipherable) prefix-code, avoiding the need for a separator to determine codeword boundaries. Huffman coding creates such a code.

C. Adaptive Huffman Coding

A different approach known as sibling property is followed [3,9] to build a Huffman tree. Here, both sender and receiver maintain dynamically changing Huffman code trees whose leaves represent characters seen so far. Initially the tree contains only the 0-node, a special node representing messages that have yet to be seen. The Huffman tree includes a counter for each symbol and the counter is updated every time when a corresponding input symbol is coded.

E. Arithmetic Coding

The Arithmetic coding Technique [5] assign an interval to each potential symbol after that a decimal number is assigned to this interval. The algorithm starts with an interval of 0.0 and 1.0. After each input symbol from the alphabet is read, the interval is subdivided into a smaller interval in proportion to the input symbol's probability. This subinterval then becomes the new interval and is divided into parts according to probability of symbols from the input alphabet. This is repeated for each and every input symbol. And, at the end, any floating point number

from the final interval uniquely determines the input data.

III. Dictionary Based Compression Techniques

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.

A. LEMPEL ZIV Algorithms Techniques

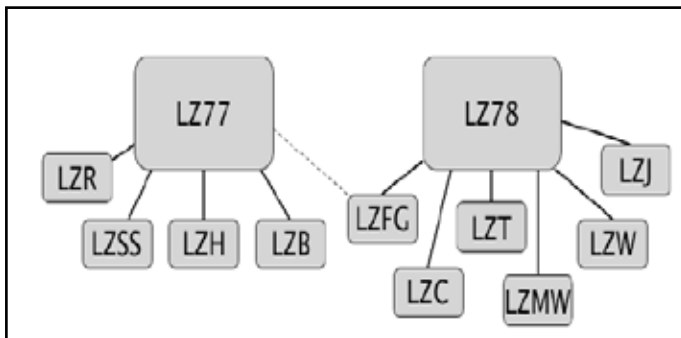


Fig 1: Family Tree of LZ Algorithms

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.

1. LZ77

Jacob Ziv and Abraham Lempel have presented their dictionary-based scheme [8] in 1977 for lossless data compression. 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.

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.

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.

IV. LZW(Lempel Ziv Welch)

Terry Welch has presented his LZW (Lempel–Ziv–Welch) algorithm [1] in 1984, which is based on LZ78. LZW was originally designed for implementation by special hardware, but it turned out to be highly suitable for efficient software implementations too. This algorithm uses the original dictionary building technique as LZ78.

LZW is a general compression algorithm capable of working on almost any type of data. LZW compression creates a table of strings commonly occurring in the data being compressed, and replaces the actual data with references into the table. The table is formed during compression at the same time at which the data is encoded and during decompression at the same time as the data is decoded.

In the original proposal of LZW, the pointer size is chosen to be 12 bits, allowing for up to 4K dictionary entries. Once the limit is reached, the dictionary becomes static.

The Lempel-Ziv-Welch (LZW) compression algorithm is widely used because it achieves an excellent compromise between compression performance and speed of execution.

A. LZW Compression Process

In this Section we will describe the actual Working of dictionary creating process of LZW Compression Algorithm. By applying LZW algorithm on the example S.

S=/WED/WE/WEE/WEB/WET

We get the following entry fields for the Dictionary creation.

Table 1: The Compression Process of LZW

S = /WED/WE/WEE/WEB/WET			
Character	Code Output	New code	New String
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T	2	/W
Total= 152 byte	Compressed size= 12 string * 12 byte =		

V. Related Work

Welch, T.A.[1], author introduces a new Compression Algorithm. The Proposed Algorithm avoids many of the problems associated

with other methods of Compression in that it dynamically adapts to the Redundancy characteristics of the data being compressed. The Effectiveness of Compression is expressed as a ratio relating character in the number of bits needed to express the message before and after Compression.

R. Nigel Horspool [2], the author describes a simple way to improve the compression without significantly degrading its speed is proposed, and experimental data shows that it works in practice and Even better results are achieved. The Lempel-Ziv-Welch (LZW) compression algorithm is widely used because it achieves an excellent compromise between compression performance and speed of execution.

Mark Daniel Ward [3], the Author investigates the Lempel-Ziv '77 data compression algorithm by considering algorithm for efficiently embedding strings in binary trees and Analysis of the multiplicity matching parameter of suffix trees was also presented.

Simrandeep kaur, et.al [4], LZW data compression algorithm is implemented by finite state machine through which the text data can be effectively compressed. The simulation results was obtained using Xilinx tools which show an improvement in lossless data compression scheme by reducing storage space to 60.25% and increasing the compression rate by 30.3%.

Nishad PM , et.al [6], the author proposed a new methodology for LZW algorithm by using binary search with insertion point and with number of dictionaries. The Proposed enhanced LZW Algorithm with multiple Dictionaries reduce the time complexity for dictionary creation.

In the proposed methodology, the single dictionary is replaced with the multiple dictionaries. Each dictionary is unique. The search in-between the dictionaries are switched based on the length of pattern in encoding as well as decoding. Each dictionary is constructed in sorted manner. To find the presents of pattern the simple binary search is implemented.

VI. Key Issues of LEMPEL ZIV WELCH(LZW) Algorithm

Even Though LZW is a Widely Used Data compression algorithms, the algorithm still posses many Limitations as well, a few of such limitations are described below.

The LZW has limited amount allocated to dictionary, i.e. 4K Locations, if this dictionary gets filled early and the input still is sparse. Here are some Scenarios LZW algorithm may take:

- LZW may simply forget about adding any more entries and use the table as is.
- It simply throws the dictionary away when it reaches a certain size.
- LZW simply throw the dictionary away when it is no longer effective at compression.
- LZW may use some schemes rebuild a string table from the last N input characters.
- Clear all entries in Dictionary and start building the dictionary again.
- Searching Complexity of Dictionary is high.

All these issues made LZW algorithm to the Direction of improvement in future.

VII. Conclusion

In this paper we have described Lempel Ziv family and various techniques associated with compression algorithms.

As discussed LZW is dictionary based algorithm, which is lossless in nature and incorporated as the standard algorithm for compression. The Lempel Ziv family has found various Applications from

text compression to multimedia but Still associated with several drawbacks which we have discussed previous.

VIII. Future Work

The Lempel Ziv Algorithm although quite old but still is the standard algorithm for various proposes ranging from text compression to image and video compression.

As discussed earlier, the LZ algorithm has various issues regarding its performance and internal structure, which various authors describes in this review tried to improve.

LZ algorithm is a Dictionary Based algorithm, but as we know the dictionary structure gets flatten out after compression process, In Future I would like to implement an LZW variant using Hash Sets and find out the performance with respect to LZ family.

References

- [1] Welch, T.A., "A Technique for High-Performance Data Compression, IEEE, "Computer", Vol. 17, No. 6, pp. 8, 19, June 1984.
- [2] R. Nigel Horspool, "Improving LZW", Dept. of Computer Science, University of Victoria P.O. Box 3055, Victoria, B.C., Canada V8W 3P6.
- [3] Mark Daniel Ward, "Exploring Data Compression via Binary Trees", Purdue University.
- [4] Simrandeep Kaur, et.al, "Design and Implementation of LZW Data Compression Algorithm", International Journal of Information Sciences and Techniques (IJIST) Vol. 2, No. 4, July 2012.
- [5] Senthil Shanmugasundaram, et.al, "A Comparative Study Of Text Compression Algorithms", International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011.
- [6] Nishad PM ,et.al, "Enhanced LZW (Lempel-Ziv-Welch) Algorithm by Binary Search with Multiple Dictionary to Reduce Time Complexity for Dictionary Creation in Encoding and Decoding", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue 3, March 2012.
- [7] Debra A. Lelewer, Daniel S. Hirschberg, Data Compression, [Online] Available: <http://www.ics.uci.edu/~dan/pubs/DataCompression.html>
- [8] Ziv, J, Lempel A., "Compression of Individual Sequences via Variable-Rate Coding", IEEE Transactions on Information Theory 24 (5), pp. 530-536, September 1978.
- [9] Md. Rubaiyat Hasan, "Data Compression using Huffman based LZW Encoding Technique", International Journal of Scientific & Engineering Research, Vol. 2, Issue 11, November-2011.