A statistics based variant of the LZW algorithm

Anshul Ranjan (05EC3013), Chitresh Bhushan (05EC3006), and Sayandeep Purkayasth (05EC3005)

***Abstract*—In this paper we propose a statistics based variant of the Lempel-Ziv-Welch algorithm, which gives a preference to the most occuring contiguous symbol sequences, while adding these sequences into a fixed size dictionary.**

***Index Terms*—LZW, compression, lossless, variant, statistics.**

I. Introduction

LEMPEL-ZIV coding is a popular lossless compression scheme and is also referred to as Lempel-Ziv-Welch (LZW) coding, following the modifications suggested by Welch for image compression. This coding scheme has been adopted in a variety of imaging file formats, such as the Graphic Interchange Format (GIF), Tagged Image File Format (TIFF) and the Portable Document Format (PDF). The basic principles of this encoding scheme are:

1) It assigns a fixed length codeword to a variable length of symbols.
2) Unlike Huffman coding and arithmetic coding, this coding scheme does not require apriori knowledge of the probabilities of the source symbols.
3) The coding is based on a dictionary or codebook containing the source symbols to be encoded and assigned codewords. The coding starts with an initial dictionary, which is enlarged with the arrival of new symbol sequences.
4) There is no need to transmit the dictionary from the encoder to the decoder. A Lempel-Ziv decoder builds an identical dictionary during the decoding process.

Inspite of its simplicity and efficiency, this method is efficient for data that needs compression when generated on the fly. However, for stored data that needs transmission after compression, a heuristic optimization is proposed.

A. The Lempel-Ziv-Welch (LZW) algorithm

The LZW algorithm consists a compressor and a decompressor. The compressor algorithm builds a translation table (dictionary) from the message being compressed. The dictionary maps fixed-length codes to sequences. The dictionary is initialized with all single-symbol sequences (256 entries in the case of 8-bit symbols). As the compressor serially examines the data stream, it stores every unique two-letter sequence into the table as a code/symbol concatenation, with the code mapping to the corresponding first symbol. As each two-symbol sequence is stored, the first symbol is sent to the output. Whenever a previously encountered sequence is read from the input, the longest such previously encountered sequence is determined, and then the code for this sequence concatenated with the extension symbol (the next symbol in the input) is stored in the table. The code for this longest previously encountered string is output and the extension symbol is used as the beginning of the next sequence.

The decompressor algorithm only requires the compressed data stream as an input, since it can build an identical dictionary from the compressed stream as it is recreating the original message. However, an abnormal case shows up whenever the sequenc symbol␣sequence␣symbol␣sequence␣symbol (with the same symbol for each symbol and sequence for each sequence) is encountered in the input and symbol␣sequence is already stored in the sequence table. When the decompressor reads the code for symbol␣sequence␣symbol in the input, it cannot resolve it because it has not yet stored this code in its table. This special case can be dealt with because the decompressor knows that the extension symbol is the previously encountered symbol.

1) Fallbacks: The LZW algorithm is a serial algorithm with no look forward. This approach is fine for data generated on the fly, needing immediate transmission with compression. However in the case of data which is available beforehand, this approach may be improvised on. Another drawback would be a potential overflow in the dictionary, in case the data length is large.

II. The variant

A. Overview

Our approach attempts to reduce statistical redundancy within the data stream. This approach considers the possibility of specific dictionary sequences occuring more often then others. We start assigning codes and adding sequences (of a particular length) to the dictionary starting with the most frequent. After adding a fixed ratio (most frequent) of the dictionary size with doublets, we move on to triplets, assign codes to them (filling another fraction of the dictionary), move on to quadruplets, and so on. We note that the original feature of LZW, in which the dictionary was created according to the order of occurence of the sequences, is lost. This asks for a transmission of the dictionary, which does not seem appealing, as it requires a lot of bits for representation. But this dictionary contains the most occuring sequences and when the data size is much larger than the dictionary size, this method will be more optimal in compression than the original LZW method.

B. Encoder Algorithm

The algorithm proposed can be divided into the following steps.

1) Extraction of different sequences of given lengths (doublets, triplets, etc.) and ordering them according to their number of occurences and building up of dictionary by

filling a fraction of the most occuring sequences and assigning fixed length bit representation.
2) Reordering of dictionary for efficient compression during its transmission.
3) Encoding the data stream with this new dictionary. Transmission of a header block, followed by transmission of the dictionary and the encoded data.

These steps are explained as below.

1) Extraction of sequences and construction of dictionary:

1) The data is scanned for all possible pairs of symbols.
2) A histogram of all such pairs is constructed.
3) The pairs are sorted in decreasing order of their occurences.
4) We take a number of the most occuring doublets (a predecided fraction of the dictionary size, based on previous trials), and add them to the dictionary after assigning fixed length bit representations for each.

The above steps are repeated for all possible sequences for increasing lengths of sequences (i.e. for triplets, then quadruplets, and so on).

2) Reordering of dictionary: The dictionary obtained so far will require a large number of bits to be transmitted if encoded as it is. Hence in order to efficiently compress it, we reorder the dictionary and transmit according to a linear predictive like encoding. This algorithm is explained as below.

1) The part of the dictionary consisting only doublets is sorted in increasing order of the first symbol of each pair.
2) This is repeated for triplets, quadruplets, etc.
3) Now, we encode this reordered dictionary a as follows. Refer Figure 2. (Note the i, jth element of the matrix a is represented as $a_{i,j}$. A_n represents the number of dictionary entries of sequence length n.
   - $a_{1,1}$ is represented using its original 8 bit representation.
   - The difference, $d_i = a_{i+1,1} - a_{i,1}$ is computed for $i = 1, 2, \ldots, A_2 - 1$. If $d_i \leq 2^D - 2$, D bits are used to encode this difference; else, $2^D - 1$ is encoded using D `1`'s, and $a_{i+1.1}$ is represented as it is (in 8 bits). Here D represents the number of bits used to represent d_i for all n-length sequences. The transmission of D `1`'s indicates that the following 8 bits represent one symbol, $a_{i+1,1}$. Note that the first column is arranged in increasing order, and hence the bit sequences are to be read as unsigned data.
   - Now, define $m_i = a_{i,j+1} - a_{i,j}$. Note m_i may be both positive or negative. And hence, we require an additional sign bit to encode m_i. Again, if $|m_i| \leq 2^{M-1} - 2$, then M bits are used to represent the m_i; else, $2^M - 1$ is encoded using M `1`'s, and $a_{i+1.1}$ is represented as it is (in 8 bits). Here M represents the number of bits used to represent m_i for all n-length sequences. The transmission of M `1`'s indicates that the following 8 bits represent one symbol, $a_{i,j+1}$.

3) Format of header block: The format is represented in Figure 3. Here

1) B_0 bits represent the number N of N-bit dictionary.
2) B_1 bits represent the maximum length, n of sequences considered in the dictionary.
3) B'_n represents the number of n-length sequences in the dictionary, where $n = 2, 3, \ldots$.
4) `Diff_bit`, represents D, as in above discussion.
5) `Mag_bit`, represents M, as in above discussion.

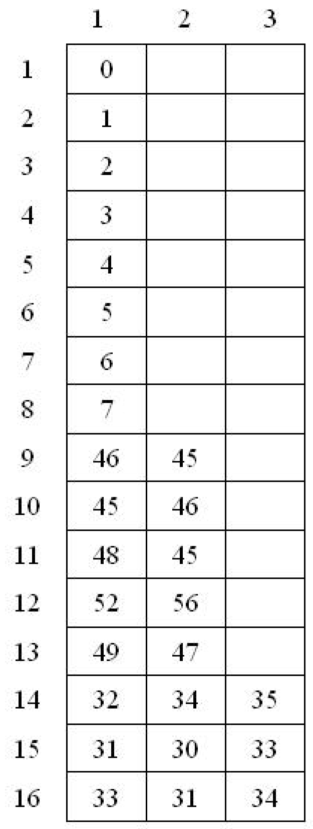
Note that B_0, B_1, B'_n's `Diff_bit`, `Mag_bit` are all of fixed length and n in B'_n is determined from B_1.

Data is encoded using simple LZW method, that is, sequences are serially replaced by the pre-assigned codeword in the dictionary constructed by the proposed algorithm.

C. Decoder algorithm

The decompression of data is quite simple, as compared to the compression procedure. The first step is to reconstruct the dictionary based on the header block recieved, and the compressed dictionary data that follows. After the dictionary is decoded, it is simply a matter of codeword subtitution by the corresponding dictionary sequence.

Fig. 1. Sample Dictionary for 3 bit data, dictionary is 4 bit.

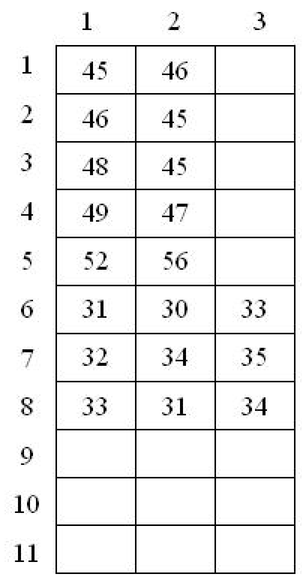


	1	2	3
1	0		
2	1		
3	2		
4	3		
5	4		
6	5		
7	6		
8	7		
9	46	45	
10	45	46	
11	48	45	
12	52	56	
13	49	47	
14	32	34	35
15	31	30	33
16	33	31	34

III. Results

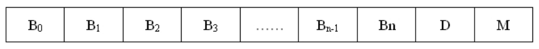
We implemented the proposed variant for upto doublets. The parameters defined in the algorithm were as follows.

Fig. 2. Reordered dictionary



	1	2	3
1	45	46	
2	46	45	
3	48	45	
4	49	47	
5	52	56	
6	31	30	33
7	32	34	35
8	33	31	34
9			
10			
11			

Fig. 3. Header block



B_0	B_1	B_2	B_3	……	B_{n-1}	Bn	D	M

Fig. 3

1) $B_0 =$ `1001`.
2) $B_1 =$ `0010`.
3) $B_2' =$ `10 0000 0000`.
4) `Diff_bit = 0010`
5) `Mag_bit` was varied across images.

Note: Images subscipted with 1, are 1/4th of the actual image, scanned column-wise. Those subscripted with 2 are the full images.

Image	No. of Bits (original)	No. of bits (LZW)	No. of bits (proposed)
$Lena_1$	32768	32364	28210
$Lena_2$	131072	138321	122782
$Tank_1$	32768	31329	29028
$Elaine_1$	32768	32526	29307
$Aerial_1$	32768	33282	31466
$Clock_1$	131072	115443	101091
$Clock_2$	524288	467361	414298

TABLE I
BITS GENERATED FOR DIFFERENT IMAGES

IV. CONCLUSION

It was observed that the proposed method achieved compression 7-10% more than the original LZW method. It was also observed that this compression was highly dependent on the content of the data source, which was as expected. Detailed images were less compressed compared to images with lesser details. Thus these results prove the robustness of the proposed method as compared to the original LZW method.

Image	Compression (LZW)	Compression (proposed)
$Lena_1$	0.98767	0.8609
$Lena_2$	1.0553	0.93675
$Tank_1$	0.95609	0.88586
$Elaine_1$	0.99261	0.89438
$Aerial_1$	1.0157	0.96027
$Clock_1$	0.88076	0.77126
$Clock_2$	0.89142	0.79021

TABLE II
COMPRESSION ACHIEVED

Fig. 4. Aerial (128x128)

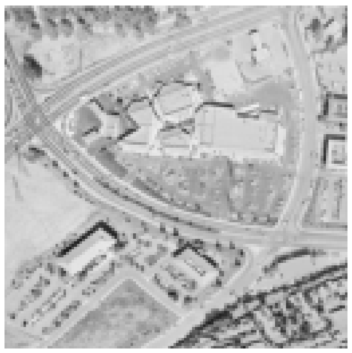


Fig. 5. Clock (256x256)

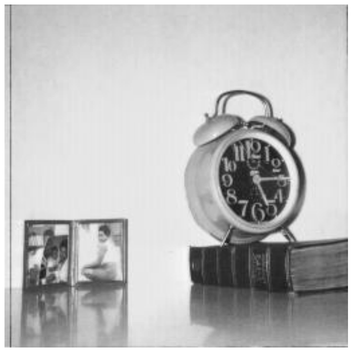
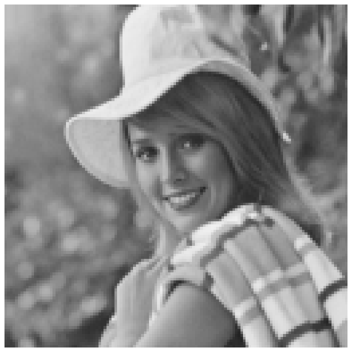


Fig. 6. Elaine (256x256)



V. FUTURE WORK

All the results were obtained only till the doublet stage. Obviously, implementation of higher length sequences would

Fig. 7. Lena (256x256)

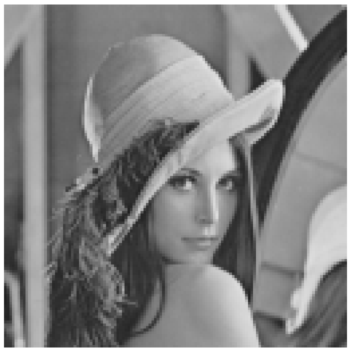
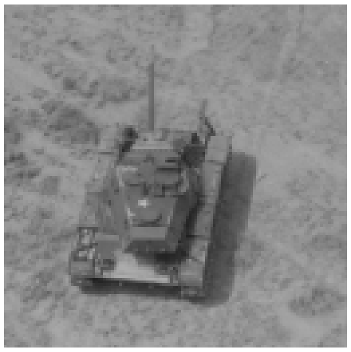


Fig. 8. Tank (256x256)



result in higher compression. Further, the method could be augmented using deletion of dictionary entries when majority of the occurences of the sequence has already been transmitted, and the dictionary could updated on the fly to include newer sequences, in its place, using slight change of protocol and algorithm.

References

[1] David Salomon, *Data Compression: The Complete Reference*, 2005, Springer, pp. 195-211.