International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-10 Issue-2, December 2020

An Approach to Efficient Dictionary Utilization and Improved Data Compression Technique for LZW Algorithm

S. Revathi, D. Thiripurasundari

Abstract*: *This paper proposes an improved data compression technique compared to existing Lempel-Ziv-Welch (LZW) algorithm. LZW is a dictionary-updation based compression technique which stores elements from the data in the form of codes and uses them when those strings recur again. When the dictionary gets full, every element in the dictionary are removed in order to update dictionary with new entry. Therefore, the conventional method doesn't consider frequently used strings and removes all the entry. This method is not an effective compression when the data to be compressed are large and when there are more frequently occurring string. This paper presents two new methods which are an improvement for the existing LZW compression algorithm. In this method, when the dictionary gets full, the elements that haven't been used earlier are removed rather than removing every element of the dictionary which happens in the existing LZW algorithm. This is achieved by adding a flag to every element of the dictionary. Whenever an element is used the flag is set high. Thus, when the dictionary gets full, the dictionary entries where the flag was set high are kept and others are discarded. In the first method, the entries are discarded abruptly, whereas in the second method the unused elements are removed once at a time. Therefore, the second method gives enough time for the nascent elements of the dictionary. These techniques all fetch similar results when data set is small. This happens due to the fact that difference in the way they handle the dictionary when it's full. Thus these improvements fetch better results only when a relatively large data is used. When all the three techniques' models were used to compare a data set with yields best case scenario, the compression ratios of conventional LZW is small compared to improved LZW method-1 and which in turn is small compared to improved LZW method-2.

Keywords* : *data compression, LZW, dictionary encoding, lossless encoding.

I. INTRODUCTION

In this current digital world data processing, data transferring and data storage is inevitable for every sector such as IT industry, banking, manufacturing, hospitals, E-commerce etc. Data being so important is collected in ways such as videos, images and text. Data which is collected massively in each and every sector requires huge amount of space for its storage. This huge amount of data will turn to a possible problem when needed to be transferred or shared from one to another. This problem can be overcome when the data is compressed. The compressed data requires low storage space comparatively.

The amount of data is directly proportional to the power it takes to compress. Along with the space and energy needed to store compressed data, the time required to compress data is another task. It is important to take time into account when data is compressed. Design of efficient algorithm is a possible solution to all kinds of the problems involved with data. It might help in decreasing the storage capacity required to store the data, reduce the power consumption and also help in saving the time required to compress the data. One such effective data compression algorithm was proposed by Abraham Lempel, Jacob Ziv and Terry Welch [1-3] and named as LZW algorithm. LZW technique represents strings with integral codes and it doesn't analyse the input text in any sorts. Rather, it just appends new characters it gets to a table called as dictionary. Compression is observed when an integral code is given as output in place of a string. LZW compression is made for files which have a lot of repetitive data. This usually happens with text and monochrome images. Files which don't have any repetitive data at all can even grow bigger due to the fact that we are replacing 8-bit character with 12-bit integral-code value. Some of the researchers have used this coding for different applications. Archarya and Mukerjee [4] have proposed a way look for the dictionary used in the form of a binary tree. All the properties of the binary tree can be easily converted into memory. When the size of the text is small the algorithm overrides the normal LZW scheme. But its functionality undermines larger texts. Samish Kamble et al., [5] provide the most explicit hardware description language (VHDL) modeling environment of the Lempel-Ziv-Welch (LZW) algorithm for binary data compression to facilitate interpretation, validation, simulation, and hardware realization. For this implementation, the LZW dictionary with all possible symbols need to be preloaded in FPGA and LZW replaces a string of compression characters with code. Agrawal Arohi et al., [6] implemented the algorithm in FPGA in which the input to the compressor is 1-bit bit stream and it is read according to the input clock cycle of the compressor. The output is an 8-bit integer stream, given in the decompressor, which is an indicator of the memory location of the bit string stored in the dictionary.

Revised Manuscript Received on December 25, 2020.

*Corresponding author

S. Revathi, School of Electronics Engineering, VIT Chennai, India, Email: revathi.s@vit.ac.in

D.Thiripurasundari*, School of Electronics Engineering, VIT Chennai, India, Email: dthiripurasundari@vit.ac.in

The reference model for data compression software using LZW is designed in MATLAB / Simulink. Simrandeep Kaur et al [7] have proposed LZW algorithm that can replace their codes with 5 bits instead of 7 bit ASCII code. The designed dictionary is based on content addressable memory (CAM) array. In this method they observed that storage space is reduced up to 60.25% and compression rate improved up to 30.3%. Yonghui Wu et al [8] have used LZ -78 improvised version of dictionary coding algorithm for hardware implementation for high-performance disk controllers. Here they focus on the problem of increasing the durability of the LZW. Compared to their previous work LZ - 77, based on the LZW additional fragments needed to maintain the involvement of the error estimators and it is also more involved. Deepa et al [9] added modified algorithm logic to assign codes to inverted string pairs. This logic uses the idea of using an existing string code with odd code for the newly encountered string LZW compression returns strings of characters with single codes. It does not perform any analysis of the incoming text. Instead, it adds each new set of characters that you see in the strings table. Dheemanth H A [10] created the dictionary when data is encoded. So you can encode on the fly. The dictionary need not be transmitted and can be built upon receiving the end on the fly. If it overflows, we need to restart the dictionary and add a bit to each code word The first 256 elements of the dictionary are given to the gray levels 0-255. Remaining part of the dictionary is gradually filled with sequences of the gray levels. LZW compression is relatively fast and gives best results when there are repetitive parts in the data. This coding technique doesn't require any prior knowledge like probability of occurrence. It's an easy encoding-decoding process with a good compression ratio. However, the disadvantage is it creates some elements in the dictionary that are never be utilised. This paper discusses on two methods of the improved LZW algorithms which utilises the dictionary in an efficient manner.

II. ALGORITHM

This section describes the encoding and decoding procedure of conventional LZW coding, and two methods of improved LZW coding.

A. Conventional LZW Coding:

A dictionary is initialized with ASCII characters coded from 0-255, in a way such that all strings can be formed using these 256 elements. The algorithm continuously scans for substrings of data till it doesn't find it in dictionary. If such a substring is seen, the code of the substring before the final iteration is taken from the dictionary and given as output, and the current substring is appended to dictionary with a newer accessible code. The previous input character is again utilised as the current initialising point to look for substrings. The encoding process is illustrated as flow chart in Fig. 1.

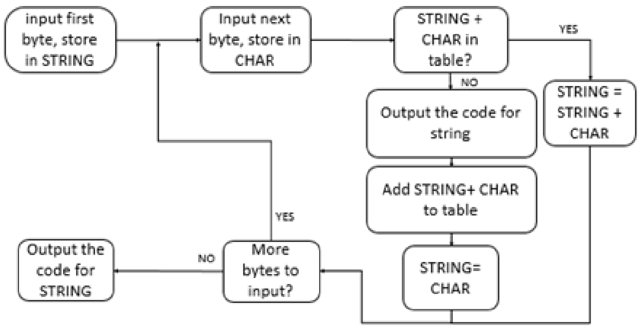
input first byte, store in STRING
Input next byte, store in CHAR
STRING + CHAR in table?
YES
NO
STRING = STRING + CHAR
Output the code for string
Add STRING+ CHAR to table
YES
NO
Output the code for STRING
More bytes to input?
STRING= CHAR

Fig. 1: Flow of conventional LZW encoding

Thus, the output of the encoding is the value from respective dictionary entry for the maximum length of the string matches. This methodology of encoding is suitable for data with repetitions. Therefore, the early snippets of the message will observe less compression and as the data increases, the compression ratio gets maximized.

At the decoder, an initial dictionary with ASCII characters coded from 0-255 is available and the dictionary update is done during the decoding process. The decoding process is carried out by taking one integral-code of the encoded output at a time and, returning the character array from dictionary which is assigned to that code. The dictionary is updated with new entry that comprises of previous decoded string and the first character of current string. If the integral-code of the encoded output is not seen in the dictionary, then the decoded output is the current sub-string appended with its first character. Also, this decoded output should be updated in the dictionary for the respective integral-code. This process of decoding is repeated until the entire coded element is complete and there are no more additions to the dictionary.

The standard dictionary address is 13 bit in size and hence the number of dictionary entries is 4096. When the dictionary is full with the maximum limit of 4096, and if a new encoded string need to be entered in the dictionary, then already existing string will be deleted and this new string will be added to dictionary. Here, the first 0-255 entries of the dictionary which are the trivial entries of 256 ASCII values will not be deleted and the deletion will start from the 257th element i.e. dictionary index of 256. The deletion will take place in sequence for every new entries of encoded string in the dictionary.

The conventional LZW algorithm is an efficient data compression technique. However, the main disadvantage of this method is the updation of dictionary once the entire index is full. In the conventional method the deletion of the dictionary entries took place in sequence from the index 256. For example, if there is 10 new encoded strings, then the existing dictionary entry from index 256 to 4095 will be erased and the new strings will be updated. The flow sequence after the dictionary filled is shown in Fig. 2.

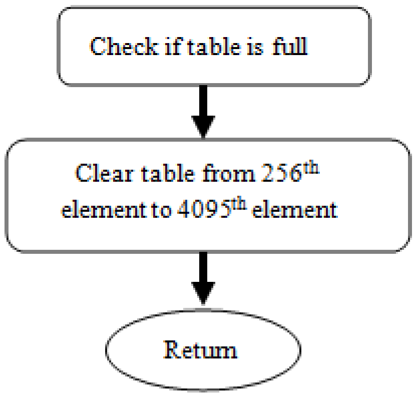

Fig. 2: Flow sequence of conventional LZW after dictionary is full

The main disadvantage of the conventional LZW is that it doesn't consider any frequently used strings and all the entries are erased. This method of dictionary updation will increase the time of encoding. Also, the repeated string will be encoded again as it doesn't appear in dictionary. The conventional LZW is good when the data set to be compressed is small or when there is less number of repeated strings.

B. Improved LZW coding – Method 1

This LZW improvement is inspired from LZMW proposed by V. Miller and M. Wegman [3]. This is based on two principles:

1. When a dictionary is completely full, the elements which are less frequently used are deleted. Many methods are there to find these elements, and the coders can find any such methods which can do the task, one of the many methods involves finding an element that has never been used to make another element, i.e. we need to find an element that has never been extended and removing the earliest such element. An independent structure must be utilised such that we can have an indication about the age of the elements and also find a way to have the information about the frequency of the elements being used. The first 256 dictionary phrases should never be deleted. It can be improved by appending to elements which have been successively found in the dictionary rather than the method used in regular LZW which appends on the first character and the current string to dictionary.

The algorithm proposed by V. Miller and M. Wegman suggested looking for the strings in dictionary which are used again. This method will increase the time of compression as every string must be compared with following strings. To overcome the disadvantage, this paper proposes a first method that considers the first principal of LZMW and implemented it in the general LZW. To improve the compression time a flag is introduced to each of the dictionary entries and when the dictionary gets filled the entries with unchanged flags are removed.

The algorithm for encoding of the improved LZW method -1 is described in Fig. 3. The flow chart depicts the working of improvement 1 and how it handles the dictionary when it's full. It sets a flag high when an element is used and thus when it has to remove elements, it only removes the ones which have the flag set as low.

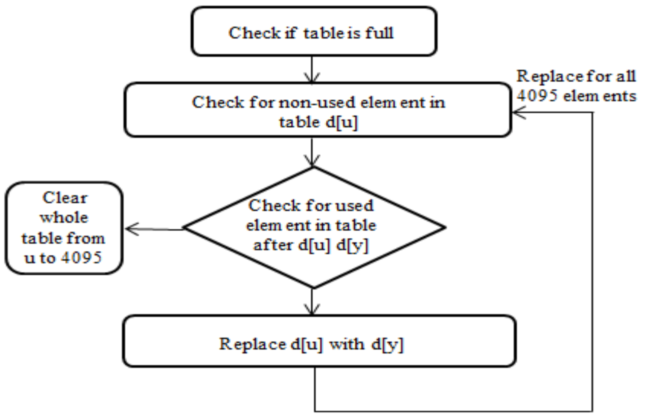
Check if table is full
Check for non-used elem ent in table d[u]
Replace for all 4095 elem ents
Clear whole table from u to 4095
Check for used elem ent in table after d[u] d[y]
Replace d[u] with d[y]

Fig. 3: Flow of improved LZW method-1

C. Improved LZW coding- Method 2

LZW improvement 1 is furthermore refined to give relatively efficient compression ratio. This algorithm is also based on removing the unused entries in the dictionary but rather than removing all the unused entries all at once, we remove them one at a time thus giving another chance for the newer entries in the dictionary. The process flow for the encoding algorithm of improved LZW method-2 after the dictionary is full is shown in Fig. 4.

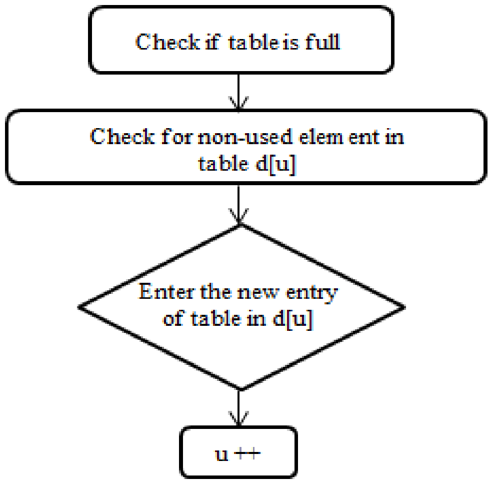
Check if table is full
Check for non-used elem ent in table d[u]
Enter the new entry of table in d[u]
u ++

Fig. 4: Flow of improved LZW method- 2

Figure 4 is the flow chart of the algorithm of improvement 2 and depicts how this technique handles the dictionary when it is full. It can be seen that the unused elements are removed only when a new element has to be added to the dictionary.

III. RESULT AND DISCUSSION

To illustrate the working of all the three algorithms and the dictionary entries, consider an input string **ABCABBABCAABCABBABCAA** with only three alphabets A, B and C. The input string was taken such that it repeats for every ten alphabet. The initial dictionary entry is given in Table I.

Table I: Initial dictionary for the given string

Index	Dictionary Content
1	A
2	B
3	C

To illustrate the effectiveness of the algorithm the dictionary is restricted to have only 10 entries. The dictionary entries at different level of encoding, status of dictionary and encoded output for conventional LZW, improved LZW method-1 and improved LZW method-2 are shown in Table II, Table III and Table IV respectively.

In the conventional LZW given in Table II, it is evident that once the dictionary is full, all the entries except the initial dictionary entry got deleted and the new encoded string are updated.

In the improved LZW method-1 given in Table III, it is evident that once the dictionary is full, all the entries except the initial dictionary entry and the flagged entry (i.e. repeated strings **'AB'** and **'CA'**) got deleted and the new encoded string are updated.

In the improved LZW method-2 given in Table IV, the unflagged entries are shown in red color. It is evident that once the dictionary is full, not all the unflagged entries are removed at once. Instead for a new encoded string, only one unflagged entry is removed and others are kept as it is. For example, when the dictionary is full and new encoded string **'ABCA'** appears, then the first unflagged entry **'BC'** in the index 5 alone is deleted and replaced by new string. This method gives enough time for the nascent elements of the dictionary and hence the compression is high.

Table II: Dictionary Entries and the encoded output of conventional LZW algorithm

Index	Conventional LZW			
	First level of dictionary updation	**Status of dictionary after full**	**Second level dictionary updation**	**Encoded output**
1	**A**	**A**	**A**	{1, 2, 3, 4, 2, 4, 6, 1, 2, 3, 4, 2, 4, 6, 1}
2	**B**	**B**	**B**	
3	**C**	**C**	**C**	
4	**AB**	-	**AB**	
5	**BC**	-	**BC**	
6	**CA**	-	**CA**	
7	**ABB**	-	**ABB**	
8	**BA**	-	**BA**	
9	**ABC**	-	**ABC**	
10	**CAA**	-	**CAA**	

Table III Dictionary Entries and the enclosed output of Improved LZW method-1 algorithm

Index	Improved LZW method-1			
	First level of dictionary updation	**Status of dictionary after full**	**Second level dictionary updation**	**Encoded output**
1	**A**	**A**	**A**	{1, 2, 3, 4, 2, 4, 6, 4, 5, 2, 2, 6, 1}
2	**B**	**B**	**B**	
3	**C**	**C**	**C**	
4	**AB**	**AB**	**AB**	
5	**BC**	**CA**	**CA**	
6	**CA**	-	**ABC**	
7	**ABB**	-	**CAB**	
8	**BA**	-	**BB**	
9	**ABC**	-	**BA**	
10	**CAA**	-	**ABCA**	

Table IV: Dictionary entries and the encoded output of improved LZW method-2 algorithm

Index	Improved LZW method-2				
	First level of dictionary updation	Status of dictionary after full	Second level of dictionary updation	Third level of dictionary updation	Encoded output
1	A	A	A	A	{1, 2, 3, 4, 2, 4, 6, 9, 7, 5}
2	B	B	B	B	
3	C	C	C	C	
4	AB	AB	AB	AB	
5	BC	BC	ABCA	ABCA	
6	CA	CA	CA	CA	
7	ABB	ABB	ABB	ABBA	
8	BA	BA	BA	BA	
9	ABC	ABC	ABC	ABC	
10	CAA	CAA	CAA	CAA	

The final dictionary content after completion of the encoding process is given in Table V.
The compression ratio for this example was calculated to have 64.25 %, 69.09 % and 76.19 % for conventional LZW, improved LZW method-1 and improved LZW method -2 respectively. It is evident from the encoded output that the compression ratio of conventional LZW is smaller compared to the improved method 1 and method 2.

Table V: Final dictionary content after encoding of string

Index	Dictionary Content		
	Conventional LZW	Improved LZW method-1	Improved LZW method-2
1	A	A	A
2	B	B	B
3	C	C	C
4	AB	AB	AB
5	BC	BC	ABCA
6	CA	ABC	CA
7	ABB	CAB	ABBA
8	BA	BB	BA
9	ABC	BA	ABC
10	CAA	ABCA	CAA

To enumerate the effectiveness of all the three algorithms, text file of different sizes were taken for compression. The size of the compressed output is given in the Table VI. The input file size of 2063 bytes, 4628 bytes, 50767 bytes and 1907036 bytes were named as A1, A2, A3 and A4 respectively. It can be seen from the Table VI that the compressed file A1 and A2 have the same size for all the three algorithms. This is because, the input file size is less than 4096 bytes and the dictionary is not yet filled and therefore, the algorithm works similar. The effectiveness of the algorithm can be found only if the dictionary is filled and this is depicted in the Table VI of input file A3 and A4. It is clear from the Table that the output file size after compression for conventional LZW is greater than LZW improvement 1, which in turn is greater than LZW improvement 2.

Table VI: Output file size in bytes for four different input file sizes

Coding/input	A1	A2	A3	A4
Conventional LZW	1968	4168	34794	1035202
LZW Improvement 1	1968	4168	32878	1017672
LZW Improvement 2	1968	4168	32281	996350

Table VII gives the compression ratio of all the three algorithms. Also, it can be seen from Table 7 that if the input file size is larger, then the compression ratio is also larger.

Table VII: The retained percentage of four different input file size after compression

Coding/input	A1	A2	A3	A4
Conventional LZW	95.4	90.1	68.5	54.3
LZW Improvement 1	95.4	90.1	64.7	53.4
LZW Improvement 2	95.4	90.1	63.6	52.2

The comparison of all the three algorithms for four different file sizes is given in the Fig. 5. Till the dictionary is filled all the three algorithms work similar. However, after the dictionary is filled the improved LZW method-2 algorithm exhibits larger compression.

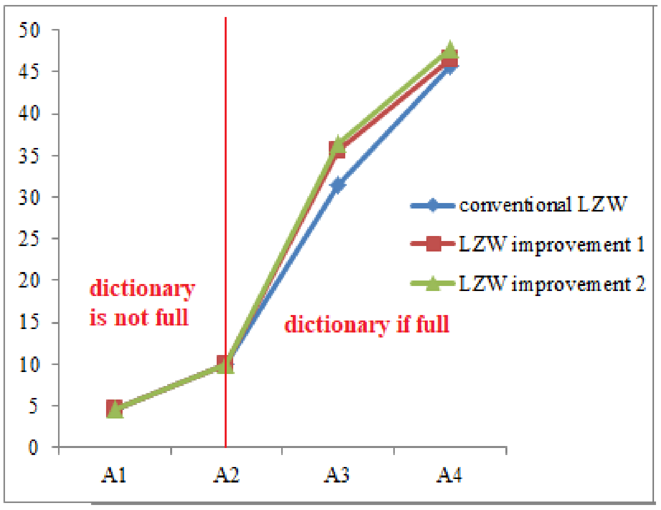

Fig. 5: Comparison of the conventional and improved LZW algorithms.

IV. CONCLUSION

In this paper the algorithm for conventional LZW was discussed with the help of flow chart and briefed on the encoding and decoding processes. Also two new improvements on the conventional LZW were suggested, and their algorithms were discussed with flow charts. An example to illustrate the working of algorithms was discussed with input string of 21 bytes and dictionary size with 10 bytes. A text file of four different size was compressed using these algorithms and the results were discussed. The file with smaller data sets gave the same results for all the three algorithms, while the file with large data sets was compressed greatly by improved algorithm.

REFERENCES

1. J. Ziv and A. Lempel, "An universal algorithm for sequential data compression", IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, 1977.
2. J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding", IEEE Transactions on Information Theory, vol. IT-24, issue 5, pp. 530-536, 1978.
3. Terry A. Welch, "A technique for high-performance data compression", IEEE Computer, vol. 17, no. 6, pp. 8-19, 1984.
4. T. Acharya, A. Mukherjee, "A Tree-based Binary Encoding of Text Using LZW Algorithm", Data Compression Conference, 1995.
5. Samish Kamble and S B Patil, “FPGA Based Data Compression using Dictionary based LZW Algorithm”, International Journal of Scientific and Engineering Research, volume 7, no. 2, February-2016, pp. 679-683.
6. Agrawal Arohi and V. S. Kulkarni, “FPGA Based Implementation of Data Compression using Dictionary based “LZW” Algorithm”, International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering, vol. 2, no. 4, 2014, pp-1391-1395.
7. Simrandeep kaur and V.Sulochana Verma, “Design and Implementation of LZW Data Compression Algorithm”, International Journal of Information Sciences and Techniques, vol.2, no.4, 2012, pp 71-81.
8. Yonghui Wu, S. Lonardi, W. Szpankowki, “Error-Resilent LZW data compression” Data compression conference (DCC’06), Snowbird, UT, 2006, pp. 193-202, doi: 10.1109/DCC.2006.33.
9. A. Deepa, Nitasha, Namrata Chopra, “Intensification of Lempel-Ziv-Welch Algorithm”, International Journal of Innovative Technology and Exploring Engineering, vol. 8, no.9, 2019, pp: 587-591.
10. Dheemanth H N, “LZW data compression”, American journal of engineering research, vol. 03, no.02, pp: 22-26.

AUTHORS PROFILE

S. Revathi is currently an Associate Professor with the School of Electronic Engineering, VIT at Chennai, Chennai. She has over 20 years of teaching and research experience. Her expertise is in CMOS, microelectronic manufacture, MEMS, microsystems, Embedded systems and IoT.

D. Thiripurasundari is professor in School of Electronics Engineering VIT Chennai with more than 27 years of teaching, research experience. Her interests are centred on the design of antennas for various applications MIMO antenna, werable, optically transparent antenna and dielctric resonatror antenna.
She had published around 30 papers in national / international journal / conference. She was also Co-PI of the project titled Design and Development of Wearable Antenna for Military applications funded by DRDO.

.

.