IMPROVING DATA COMPRESSION RATIO BY THE USE OF OPTIMALITY OF LZW & ADAPTIVE HUFFMAN ALGORITHM (OLZWH)

Pooja Jain , Anurag jain and Chetan Agrawal

Computer Science Department, Radharaman Institute of Technology and Science, Bhopal

ABSTRACT

There are several data compression Techniques available which are used for Efficient transmission and storage of the data With less memory space. In this paper, we have Proposed a two stage data compression Algorithm that is OLZWH which uses the Optimality of Lempel- Ziv-Welch (OLZW) and Adaptive Huffman Coding. With this proposed Algorithm, the data Compression ratios are Compared with existing Compression Algorithm for different data sizes.

KEYWORS

Optimality of Lempel-Ziv-Welch (OLZW), Adaptive Huffman coding, Compression Ratio.

1. INTRODUCTION

As we see any type of data files of any size Character By character, we notice that there are Many recurrence pattern exhibits in it. a data Compression [2] [6] technique takes the advantage Of such repetitive sequence of data in order to Provide a potential cost saving associated with Transmitting less amount of data, reduces storage

Requirement and reduces the probability of transmission errors. A data compression Techniques [4] are divided into two main classes. That are (i) Lossless data Compression and (ii) Lossy data compression In lossless data compression, the compression Process is carried out without loss of data or Information during compression. Examples of such Lossless compression includes A) Statistical data compression Techniques (i) Huffman coding (ii) Run Length Coding B) Dictionary based data compression techniques (i) Lempel- Ziv-Welch (LZW) Huffman coding is a general compression technique in Which data in term of symbol is based on their statistically Occurred frequencies also called probabilities. The Symbols that arise more commonly are assigned a smaller bit whereas the symbol that occur less frequently Are assigned longer bit. in 1977, Abraham Lempel and Jacob Ziv invented a new coding technique for data Compression (LZ77) which uses dictionary for encoding the data Further revised by Welch in 1978 called LZW78.The LZW algorithm is an adaptive dictionary-based Approach that builds a dictionary based on the Document That is being compressed. The LZW begins with An

initial dictionary includes all the symbols in the Alphabet, that builds the Dictionary by adding new phrases in the dictionary. As it come across new symbols in the text which is being compressed. The LZW [1][3][6] compression technique has certain limitations which reduces the compression ratios that are as follows:

(i) The dictionary is initialises for all the characters symbols in at the beginning of the encoding process Some of the dictionary may not be used during encoding of the data files but that unused character symbols occupy some code values that cannot be used for other necessary phrases of dictionary.
(ii) LZW started with 256 encoding code values for 256 character symbols that are 8bit long. The character symbol beyond 256 has to be encoded with 9 bits. Thus, overcome above limitations OLZW [4] [5] compression technique is used.

In OLZW data compression technique, the encoding procedure started with an empty dictionary. The character symbols to be encoded are assigned codes starting from '1' in the dictionary and the character symbol to be encoded whose index not present in the dictionary is encoded with 8bit ASCII code and inserted into dictionary as a new code. An adaptive Huffman [1][6] coding also called Faller Galler –Knuth algorithm uses the defined word string which determine the Mapping from source message to code word based on a running estimate of the probabilities to the source message. The code is adaptive and changes so as to stay on optimal for the current estimates. An advantage using the Adaptive Huffman coding is that it uses only one pass for compression and provides better performance than static Huffman coding.

2. OLZWH (Proposed Algorithm)

In the proposed two stage data compression algorithm (OLZWH), dictionaries formed for input character symbols of OLZW is considered into two segment that are (i) Set of indices, and (ii) Set of ASCII code Proposed algorithm starts with usual OLZW compression process and whenever there is occurrence of ASCII code, Adaptive Huffman algorithm is applied to it. The algorithms for OLZWH and flowchart for compression in Fig.1) & decompression in Fig(2) shown below

The flow for the same is shown as below-

(i) Compression Algorithm:

```
Start
Initialize TEMP_BUFF as ByteArray;
Start with empty dictionary and next code=1, PHARSE=NULL,
Flag=0;
Read first element from input file/stream as Ch;
Store 1 bit flag with 8 bits ASCII code for Ch to TEMP_BUFF;
Append Ch to dictionary with code value as next_code;
Next_code is incremented by one;
While (not end of file/stream) do
        Read next element from input file/stream as Ch;
        If (concatenation of PHARSE and Ch is in dictionary),
then
                Flag=1; PHARSE = concatenation of PHARSE
and Ch;
         Else
If (dictionary is full)
                Remove all phrase from dictionary;
End if;
       Append PHRASE to dictionary with code value as
next_code;
        next_code is incremented by one;
```

```
If (Flag=0), then
Store 1 bit flag to output file / stream and
8 bits ASCII code for Ch to TEMP_BUFF;
PHRASE=NULL;
Else
CL=Find_code_length (next_code-1);
Store flag with CL bits code for Ch from dictionary to
TEMP_BUFF;
If (Ch is not in dictionary), then
Flag=0;
Store 1 bit flag to output file / stream and
8 bits ASCII code for Ch to TEMP_BUFF
If (dictionary is full)
Remove all phrase from dictionary;
End if;
Append Ch to dictionary with code value as next_code;
next_code is incremented by one;
Else
PHRASE=Ch;
Flag=1;
End if;
End if;
End while;
If (Flag=1), then
CL= find_code_length (next_code-1);
Store flag with CL bits code for PHRASE from dictionary to output
file / stream;
End if;
Initalize COMP_BUFF as ByteArray;
COMP_BUFF= CALL AdaptiveHuffman (TEMP_BUFF)
If (COMP_BUFF.length < TEMP_BUFF) then
        Store 1bit flag with COMP_BUFF to the starting of output
file / stream;
Else
        Store 1bit flag with TEMP_BUFF to the starting of output
file / stream;
End if;
END
```

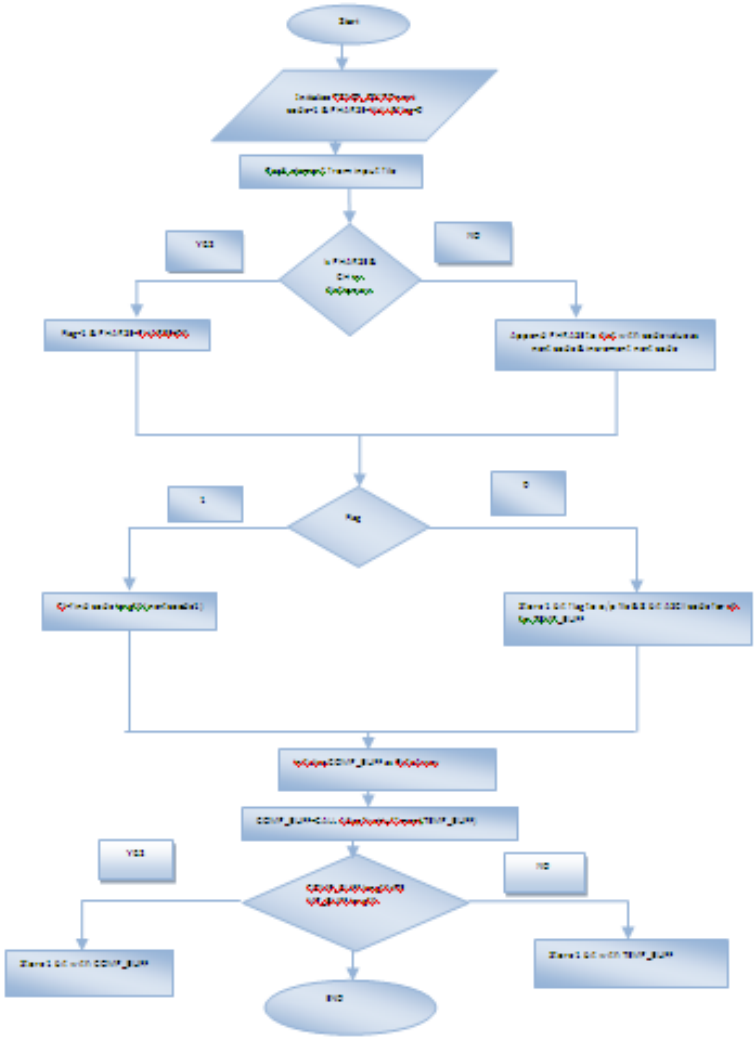

Figure 1 Flow Chart for OLZWH Compression Algorithm

(ii) Decompression Algorithm:

```
START
Initialize TEMP_BUFF as ByteArray;
Initialize COMP_BUFF as ByteArray;
Read first one bit from compressed file/stream as Flag.
If (Flag=1), then
        Read compressed data into COMP_BUFF;
        TEMP_BUFF= CALL
AdaptiveHuffman.decompress(COMP_BUFF);
Else
        Read uncompressed data directly into TEMP_BUFF;
End if;
Start with empty dictionary and next_code=1,
PHRASE=NULL;
Read first one bit from compressed file/stream as Flag.
While (not end of compressed file/stream) do
If (Flag=0), then
Read next 8 bits from TEMP_BUFF as Ch;
Append concatenation of PHRASE and Ch to the dictionary
with code value as next_code;
Increment next_code by one;
Store Ch to the output file / stream;
If (Ch is not in dictionary), then
Append Ch to dictionary with code value as next_code;
Increment next_code by one;
End if;
PHRASE=NULL;
```

```
P=0;
Else
CL= find_code_length (next_code-1);
Read next 8 bits from TEMP_BUFF as Ch;
Y= translation of Ch;
Store phrase Y to the output file / stream;
If (P=1), then
Append concatenation of PHRASE and Ch to dictionary
with code value as next_code;
Increment next_code by one;
End if;
PHRASE=Y;
P=1;
End if;
Read next one bit from compressed file/stream as Flag;
End while;
END
```

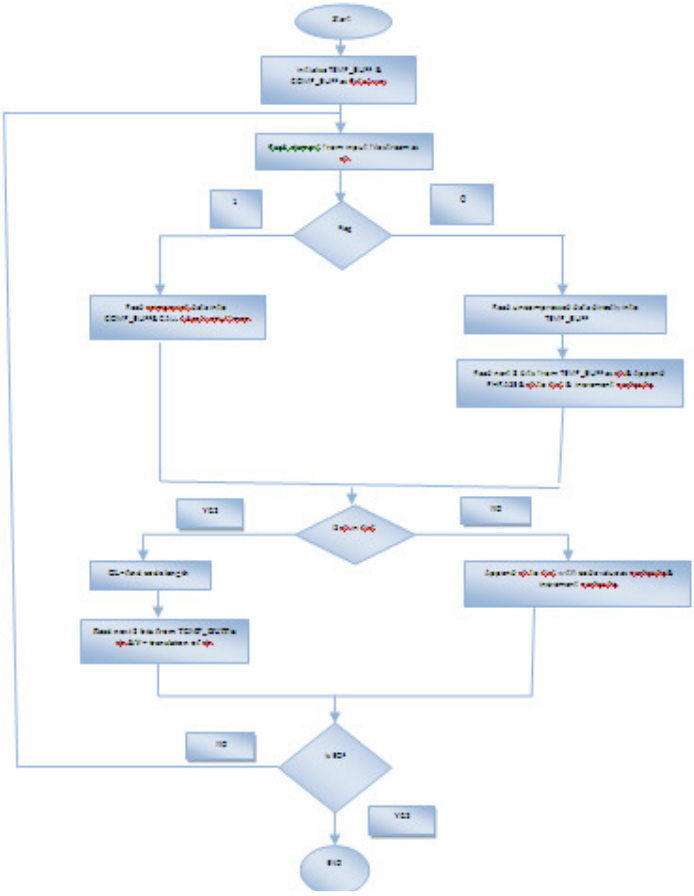

Figure 2 Flow Chart for OLZWH decompression Algorithm

3. OBSERVATIONS & RESULTS

To implement this algorithm and for comparing it with other compression techniques like LZW-15, OLZW, we calculated compression ratios in Percentages (%) for different data files of different sizes are taken. Input data files are shown in table (A) and the chart to show compression ratio in percentage (%) is shown in (Fig. 3).

Table – A. Input data files

File Name	File Size in Bytes	LZW-15	OLZW	OLZWH
e-mail.doc	58368	51.51	58.83	60.06
Brain-fingerprint.doc	81408	17.35	31.02	33.3
Capcha.doc	219648	1.81	18.96	21.81
Imageprocessing.doc	411136	9.52	25.11	29.33
404-1.html	1358	26.8	29.23	31.15
RSSFeeds.html	10012	56.09	62.22	66.53
Clickme.html	41385	27.67	39.1	45.82
LZWcompression-Rossett.html	385801	56.27	64.18	77.01
Limiteddic.java	274	16.79	18.61	20.44
ByteArray.java	2354	35.51	45.84	47.62
LZW.java	3706	32.68	40.66	46.41
Compactinputstream.java	5242	30.71	39.93	48.42
About_continue.help.txt	1185	39.66	40.68	41.86
Default.help.txt	2223	38.01	45.66	46.87
About_eventslog.help.txt	5637	40.29	48.99	54.55
Appv.instlog.txt	1013458	49.31	57.97	80.3

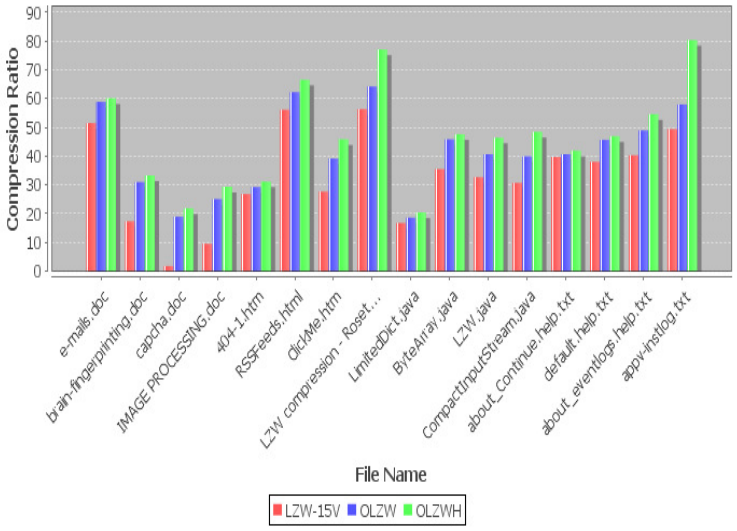

Figure 3. Compression ratio in percentage (%)

4. CONCLUSION

Thus, the proposed algorithm proves to have better compression ratio for different files as well as different file size than other available data compression techniques. Limitation of OLZW of providing poor compression ratio for larger size is also eliminated. The proposed OLZW technique eliminates some of the problems of the LZW coding and enhances the performance of the same. The proposed technique works very well for particularly small size files most of the time than two versions of LZW (i.e. LZW12 and LZW15V). And the performances of OLZW are not so poor for large size files also. It offers better compressions for large size files than LZW12 most of the time. The compression rates of OLZW for large size files are not as well as LZW15V, but close to it. The proposed technique has a great scope of modifications to make it suitable for large size files also by populating the dictionary with combination of character of phrase instead of the phrase itself and removing phrases not used for longest period of time if the dictionary gets full. It can also be used for image compression.

Dictionary Based Compression Algorithm is found to be more better than LZW in terms of file size reduction. Dictionary Based Compression is better in large size text file and it is similar to LZW Compression file when the file size is small. Dictionary Based Compression algorithm is to search a pattern in the compressed file without getting decompressed to it. Hence reducing the searching time because of being done it in compressed file. So, it would be proved as an optimal searching concept for pattern searching in the compressed text files.

REFERENCES

[1] Raja P. And Saraswathi,"An effective two stage Text compression & decompression technique For data compression " International journal of Electronics & communication engineering (ISSN 0974-2166) vol.4 no.29,2011

[2] Ming-Bo lin, jang-feng Lee& gene Eu jan, "A lossless data compression & decompression Algorithm and its hardware Architecture"IEEE Transaction on September 2006.

[3] Haroon Altarownesh ,Mohmmad Altarownesh "A data compression technique on text files, A comparison study" International journal of Computer application .(0975-8887) vol.26, No.5,july 2011.

[4] Nishad P.M., Monicka chezian , " Optimization (Lempel Ziv Welch) algorithm to reduce time Complexity for dictionary creation in encoding & decoding" AJCSIT (ISSN 2249-5126)

[5] Utpal Nandi, Jyotsna Kumari 'Modified Compression technique based on Optimality of LZW code" Elsevier 2013

[6] MIT 6.02 Draft lecture notes Chapter 3, Compression Algorithm: Huffman & Lempel Ziv Welch.