*International Journal of Engineering & Technology*, 7 (2.21) (2018) 127-130

**International Journal of Engineering & Technology**

*Website: www.sciencepubco.com/index.php/IJET*

***Research paper***

# Performance optimization of dual stage algorithm for lossless data compression and decompression

**Shrikanth Shirakol[1*], Akshata Koparde[2], Sandhya[3], Shravan Kulkarni[4], Yogesh Kini[5]**

*[1]Assistant Professor, Department of Electronics and Communication Engineering,*
*SDM College of Engineering and Technology, Dharwad, India.*
*[2]Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, India.*
*[3]Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, India.*
*[4]Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, India.*
*[5]Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, India.*
**Corresponding author E-mail: shrikanthks09@gmail.com*

**Abstract**

In this paper, an optimized dual stage architecture is proposed which is the combination of Lempel-Ziv-Welch (LZW) Algorithm at the first phase and Arithmetic Coding being the later part of Architecture. LZW Algorithm is a lossless compression algorithm and code here for each character is available in the dictionary which reduces 5-bits per cycle as compared to ASCII. In arithmetic coding the numbers are represented by an interval of real numbers from zero to one according to their probabilities. It is an entropy coding and is lossless in nature. The text information is allowed to pass through the proposed architecture and it gets compressed to the higher rate.

***Keywords:*** *LZW encoding technique, arithmetic encoding technique, and proposed dual stage architecture for Lossless Data Compression and Decompression*.

## 1. Introduction

Referring to modern trends, the web users' methodology demand a small sized source with high information rendering capacity. This latest trend is observed and inspired in optimizing a proposed Dual Stage Architecture for Lossless Data Compression and Decompression [2]. Data Compression is a method of reducing the number of input characters before storing or transmitting the information. Data Compression is broadly classified as Lossless and Lossy Compression. Lossy type of Compression finds its application in Compression of images. Lossless Compression is a key in text or binary file compression for storing and transmission. The dual stage architecture comprises of compression techniques which are lossless in nature and can reduce the input characters in a very substantial way. Both the algorithms are adaptive and there is no need of transferring any extra information from encoder to decoder. The first stage is dictionary base ASCII compared procedure which is LZW Algorithm [1]. Each character has a representation in terms of index number in dictionary. That is the LZW algorithm constructs its own dictionary by concatenating the input characters and replaces each repetition of string by index of the dictionary that is being created. In a similar manner the decoder builds a separate dictionary based on decoding algorithm, thus original text data is retrieved. The second stage of the proposed architecture is the Arithmetic Coding. This is a Variable Length Coding scheme and encodes each character or numerical constants separately. It requires a prior knowledge of the symbol probabilities and accumulates the number of repetition (frequency) of the characters or numerical constants of the input data. Transferring of each characters is done using variable number of bits based on its frequency. The principles used by these two algorithms may be different but they can be combined into a cascaded algorithm which may yield higher compression ratio and being appropriate for storage and communication purpose. In this paper the performance optimization for lossless data compression is done by cascading these two algorithms. Here the frequency of each entries of newly build LZW dictionary is accumulated during compression in the first stage and the same frequencies are used by Arithmetic coding in the next stage of compression. Since the decimal output codes generated from LZW encoder differ in their frequencies an extra compression is being achieved in the second stage. This yields in better compression ratio in addition to primary data compression by the LZW algorithm.

## 2. Related work

Proposed architecture has two stages for data compression and decompression. The two stages of the architecture contains two separate algorithms namely LZW algorithm and arithmetic coding [2].

### LZW algorithm as a first stage

Lempel-Ziv-Welch is a very simple yet effective algorithm. This versatile algorithm uses a code table which has a choice of providing 4096 entries in table. When this algorithm starts to encode, the code table has 256 entries with next table codes being empty. As encoding continues, the algorithm catches repeated sequence in data and adds them to the code table. The key point in the process being the input file is not added to code table until it has been placed in the compressed file as separate character. The dictionary that is generated need not be stored with compressed data. The compressor and de-compressor sees the dictionary with

256 single character sequences corresponding to ASCII character set.

This algorithm is based on code table initially consisting of 256 ASCII entries from 0 to 255. The characters given as input get concatenated and is compared to the code table. If it is not found, then the code table gets updated by adding the concatenated string from the entry of 256. In the first entry the ASCII value of the input character is the output code. Later as concatenated string is repeated, it gets updated code value(12 bits) rather than individual ASCII value.

### Flow chart for LZW data compression

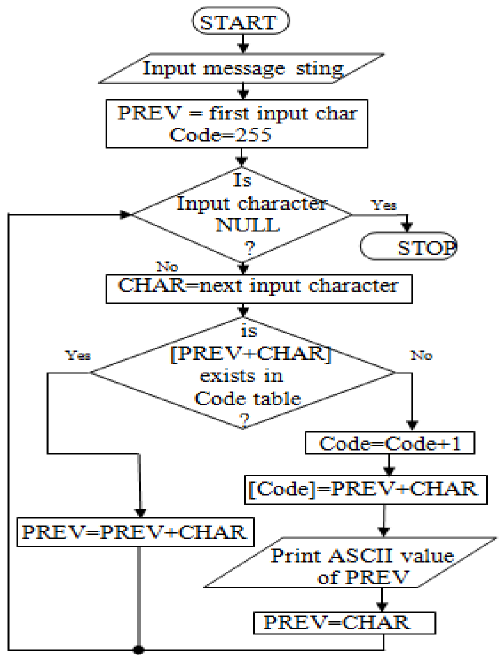


**Fig.1:** Flow chart of LZW data compression

### Flow chart for LZW data decompression

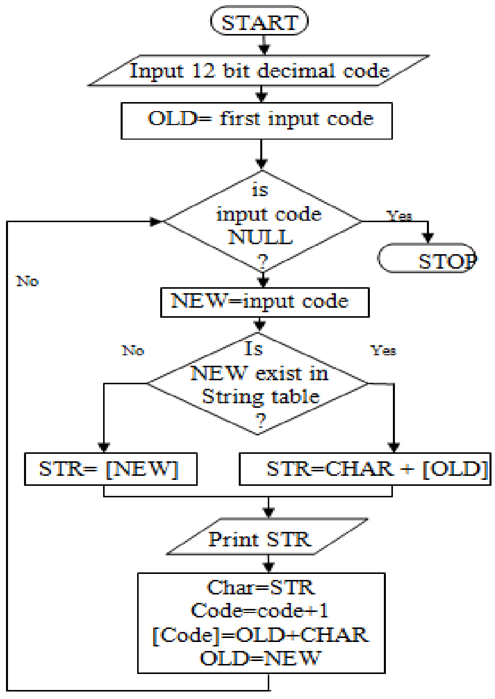


**Fig.2:** Flow chart of LZW decompression

An example illustrating the LZW Compression. INPUT string: **ABABBAA**

**Table I:** Example for LZW Compression

| PREV | CHAR | PREV +CHAR | Code Table | | **Output** |
|---|---|---|---|---|---|
| | | | New Code | PREV + CHAR | |
| A | B | AB | 256 | AB | **65** |
| B | A | BA | 257 | BA | **66** |
| A | B | AB | - | - | **-** |
| AB | B | ABB | 258 | ABB | **256** |
| B | A | BA | - | - | **-** |
| BA | A | BAA | 259 | BAA | **257** |
| A | - | - | - | - | **65** |

The output decimal code: **65, 66, 256, 257, 65.**

An example illustrating the LZW decompression. Decoding the code which was generated in LZW Compression example

INPUT Code: **65, 66, 256, 257, 65.**

**Table 2:** Example for LZW Decompression

| OLD | NEW | STR | **Output** | CHAR | STRING TABLE | |
|---|---|---|---|---|---|---|
| | | | | | New Code | OLD + CHAR |
| 65 | - | - | **A** | A | - | - |
| 65 | 66 | B | **B** | B | 256 | AB |
| 66 | 256 | AB | **AB** | A | 257 | BA |
| 256 | 257 | BA | **BA** | B | 258 | ABB |
| 257 | 65 | A | **A** | A | 259 | BAA |

The output string is: **ABABBAA.**

The generated string is same as the input string given in the example of LZW Compression. There is no loss in any of the character of message string. This is a lossless data Compression.

## Arithmetic coding

In arithmetic coding the numbers are represented by an interval of the real numbers from zero to one according to their probabilities. It is an entropy coding and a lossless compression[3]. The input message is given a unique tag value and is calculated by the probability. Arithmetic coding assigns variable length codes to a variable group of symbols. All the symbols in a message or a code are considered together to assign a single arithmetic code word. There is no one-to-one correspondence between the symbol and its corresponding code word. This is very efficient coding method in which codes are compressed to a larger extent and it helps to attain greater compression.

### Flow chart for arithmetic encoding

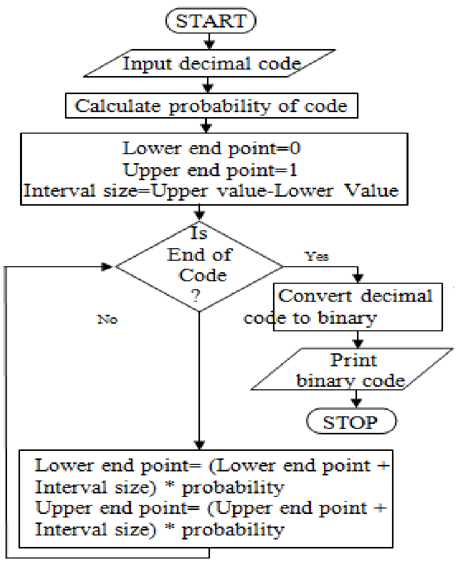

START
Input decimal code
Calculate probability of code
Lower end point=0
Upper end point=1
Interval size=Upper value-Lower Value
Is
End of
Code
?
Yes
Convert decimal
code to binary
Print
binary code
STOP
No
Lower end point= (Lower end point +
Interval size) * probability
Upper end point= (Upper end point +
Interval size) * probability

**Fig.3:** Flow chart of Arithmetic encoding

**Flow chart for arithmetic decoding**

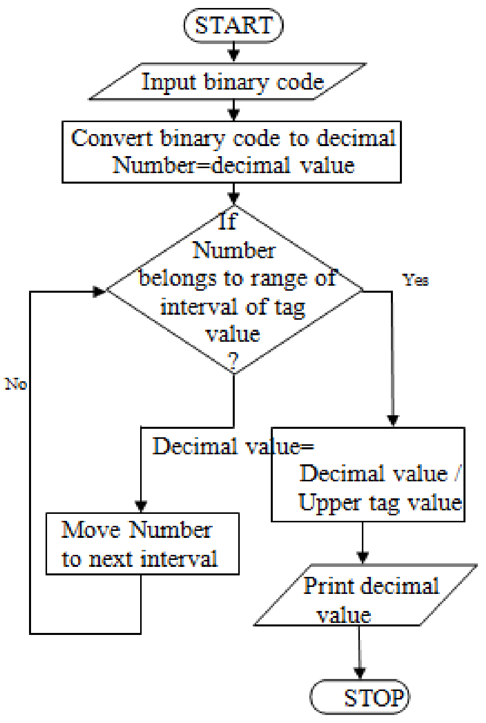

START
Input binary code
Convert binary code to decimal Number=decimal value
If Number belongs to range of interval of tag value ?
Yes
No
Decimal value= Decimal value / Upper tag value
Move Number to next interval
Print decimal value
STOP

**Fig.4:** Flow chart of arithmetic decoding

An example illustrating the Arithmetic encoding. Encoding the codes 1, 2and 3 repeating with frequencies 2, 4 and 10 respectively. From the given data: Source: 1, 2, 3withfrequencies: 2, 4, 10

Calculation: Probabilities: P(1) = 2/16 = 0.125, P(2) = 4/16 = 0.25 and P(3) =10/16 = 0.625

The upper and lower end points at each stage is calculated using mathematical formulae:

$$\boldsymbol{u^{(n)}} = \boldsymbol{l^{(n-1)}} + \left[\boldsymbol{u^{(n-1)}} - \boldsymbol{l^{(n-1)}}\right] \boldsymbol{F_X(X_n)} \qquad (1)$$

$$\boldsymbol{l^{(n)}} = \boldsymbol{l^{(n-1)}} + \left[\boldsymbol{u^{(n-1)}} - \boldsymbol{l^{(n-1)}}\right] \boldsymbol{F_X(X_n - 1)} \qquad (2)$$

The calculations are done as per the algorithms and above mentioned formulae.

- When 1 is transmitted

Lower limit=0, Upper limit=0.125
0.125-0=0.125 =>0.125*0.125=0.015625
=>0.015625+0=0.015625

- When 2 is transmitted

Lower limit=0.015625, Upper limit=0.046875
0.046875-0.015625=0.03125 =>0.03125*0.125=0.00390625
=>0.00390625+0.015625=0.01953125

- When 3 is transmitted

Lower limit=0.02734375, Upper limit=0.04687

- Taking average of these two values we get 0.037109375.
- Converting 0.037109375 to binary form: 00001011

Hence,
The output code is: **0 0 0 0 1 0 1 1**

An example illustrating the Arithmetic decoding. Decoding the code which was generated in Arithmetic encoding example.
INPUT code: **0 0 0 0 1 0 1 1**

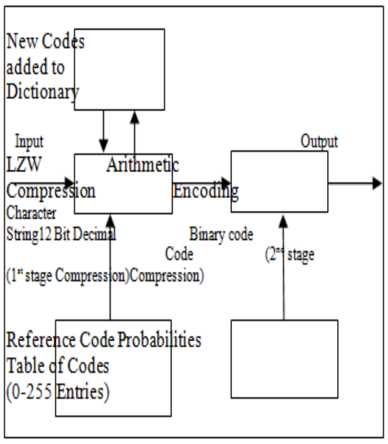
- Converting from binary to decimal = 0.037109375
- Initial value is 0.037109375
  0.037109375 is in between the tag values [0, 0.125].
  Then the transmitted values is **1**.
- Dividing 0.037109375 by 0.125=0.296875
  0.296875 is in between the tag values [0.125, 0.375].
  Then the transmitted value is 2.
- Dividing 0.296875 by 0.375=0.7916666
  0.7916666 is in between the tag values [0.375, 1].
  Then the transmitted value is 3.

The output Code has Source values: **1, 2, 3.**

The original code is retrieved from arithmetic decoding and also larger compression is achieved in encoding technique.

## 3. Proposed two stage algorithm

The character string is fed as an input to the architecture at the first stage being the LZW that includes the existing code table having a reference of the ASCII characters from 0 to 255. The new codes are added to the dictionary which will be used if the character string input has repetition of previous characters. The output from the first stage is compressed to 12 bits decimal code and serves as an input for the second stage that is Arithmetic coding. The key being the probability of codes repeated in a regular interval of time. The second stage output which is the final output of the architecture will be binary coded data marking higher compression and easy for storage and transmission. Since the decimal output codes generated from LZW encoder differ in their frequencies an extra compression is being achieved in the second stage. This yields in better compression ratio in addition to primary data compression by the LZW algorithm.



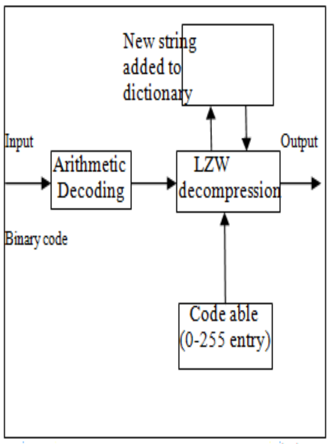


**Fig.5:** Design flow of encoder for proposed architecture
**Fig.6:** Design flow of decoder for proposed architecture

## 4. Performance analysis

Simulation is done on MATLAB-R2014b tool with different number of characters as input and their corresponding compression ratio is calculated for both LZW algorithm and proposed two stage algorithm. The result are as follows.

**Table 3:** Comparison of Data Compression at LZW and LZW+Arithmetic Algorithms

| Input (Number of bites) | LZW | | LZW + Arithmetic | |
|---|---|---|---|---|
| | Outp--ut bites | Compres-sed bites | Outp-ut bites | Compress-ed bites |
| 36 | 24 | 12 | 16 | 20 |
| 72 | 28 | 44 | 20 | 52 |
| 152 | 64 | 88 | 53 | 99 |

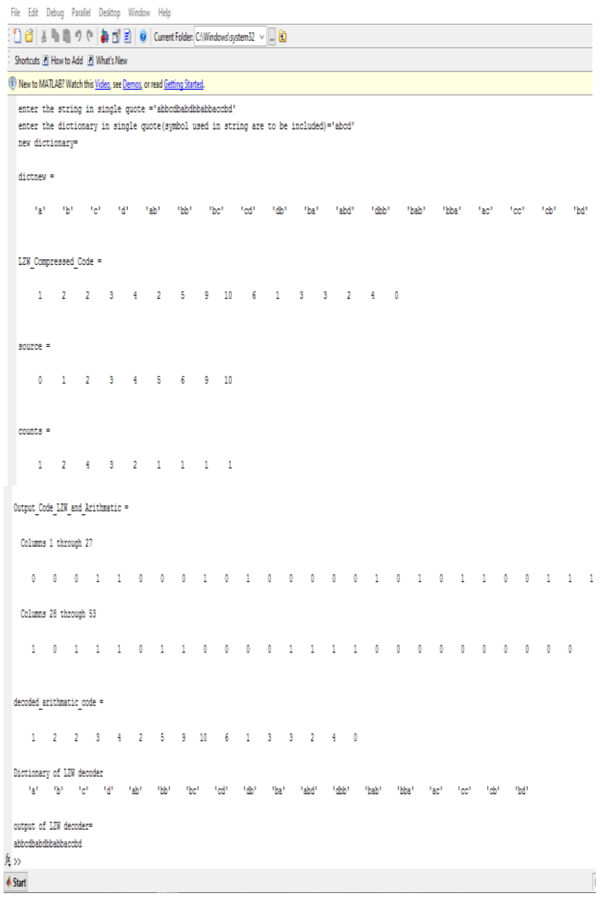
**Table 4:**Comparision of data Compression ratios between lzw and LZW+Arithmetic Algorithms

| **Input (Number of bits)** | **Compression ratio at LZW algorithm** | **Compression ratio LZW + Arithmetic algorithm** |
|---|---|---|
| **36** | **33.33%** | **55.55%** |
| **72** | **61.11%** | **72.22%** |
| **152** | **57.89%** | **65.13%** |

A survey on lossless data compression gives the result as nearly 30% of data gets compressed through LZW algorithm [4] and nearly 52% data compression can be obtained by cascading LZW and Huffman algorithms [8]. In this regard it gives better compression rate on cascading LZW and Arithmetic algorithms.

## 5. Algorithm simulation

To evaluate the compression performance of proposed algorithm a MATLAB coding is done by cascading LZW algorithm and Arithmetic Coding. Simulation results are obtained in MATLAB-R2014b. The simulation result is as follows,


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Windows\system32
Shortcuts How to Add What's New
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Start

```
enter the string in single quote ='abbcdbabdbbabbaccbd'
enter the dictionary in single quote(symbol used in string are to be included)='abcd'
new dictionary=

dictnew =

    'a'    'b'    'c'    'd'    'ab'    'bb'    'bc'    'cd'    'db'    'ba'    'abd'    'dbb'    'bab'    'bba'    'ac'    'cc'    'cb'    'bd'


LZW_Compressed_Code =

     1     2     2     3     4     2     5     9    10     6     1     3     3     2     4     0


source =

     0     1     2     3     4     5     6     9    10


counts =

     1     2     4     3     2     1     1     1     1

Output_Code_LZW_and_Arithmatic =

  Columns 1 through 27

     0     0     0     1     1     0     0     0     1     0     1     0     0     0     0     0     1     0     1     0     1     1     0     0     1     1     1

  Columns 28 through 53

     1     0     1     1     1     0     1     1     0     0     0     0     1     1     1     1     0     0     0     0     0     0     0     0     0     0


decoded_arithmatic_code =

     1     2     2     3     4     2     5     9    10     6     1     3     3     2     4     0

Dictionary of LZW decoder
    'a'    'b'    'c'    'd'    'ab'    'bb'    'bc'    'cd'    'db'    'ba'    'abd'    'dbb'    'bab'    'bba'    'ac'    'cc'    'cb'    'bd'

output of LZW decoder=
abbcdbabdbbabbaccbd
>>
```

## 6. Conclusion

The dual stage architecture has a significant compression ratio and the compressed data is loss-less in nature making it one of the key factor for the application in the field of text and data compression. The dual process in the proposed architecture compresses data on an average of **65%**.

## Future work

The dual compression stage presented in the proposed paper is mode based design. The future scope includes the dual stages to be implemented using very large scale integration (VLSI) methodology which is faster in terms of processing.

## Acknowledgment

We sincerely thank SDM College of Engineering and Technology for the humble support. We also extend our gratitude to Associate Prof. Savitri Raju (Dept. of ECE SDMCET, Dharwad) for her continuous support. Working as a group for this reason was a learning experience and will be a memorable moment in our lives.

## References

[1] Ming-Bo, L, Member, IEEE, Jang-Feng L, & Gene Eu J, "A Lossless Data Compression And Decompression Algorithm And Its Hardware Architecture", *IEEE Transaction on Very Large Scale Integration(VLSI) systems*, Vol.14, No.9, (2006).

[2] Yehoshua P, Venkat M & Nageshwar K, "The Cascading Of The Lzw Algorithm with Arithmetic Coding", *IEEE journal* (1991).

[3] Redinbo GR., "Protecting Data Compression: Arithmetic Coding, *IEE Proceedings-Computers and Digital Techniques*, Vol.147, No.4, (2000).

[4] Simrandeep K & Sulochana Verma V, "Design and Implementation of Lzw Data Compression Algorithm", *International Journal Of Information Sciences And Techniques (IJIST)* Vol.2, No.4, (2012).

[5] Behrouz F, 15th Chapter, Data Compression, Foundations of Computer Science Cengage Learning.

[6] Jung B & Burleson WP, "Efficient VLSI for Lempel-Ziv Compression in Wireless Data Communication Networks", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst*., Vol.6, No.3, (1998), pp. 475–483.

[7] Zhu C & Xu H, "Design and Implementation Of Lossless Compression Encoding For High-Speed Data Acquisition And Storage", *IEEE 12th International Conference on Electronic Measurement & Instruments*, (2015).

[8] Sandhya Shravan K & Yogesh K, "Pre Equal Architecture For Lossless Data Compression And Decompression Using Hybrid Algorithm", *International Journal Of Advanced Electrical and Electronic Engineering (IJAEEE)*, Vol.6 No.1, (2017).

[9] Welch TA, "A Technique for High-Performance Data Compression", *IEEE Comput.,* Vol.17, No.6, (1984), pp.8–19.

[10] Ranganathan N & Henriques S, "High-Speed Vlsi Designs For Lempel-Ziv-Based Data Compression", *IEEE Trans. Circuits Syst. II. Analog Digit. Signal Process.,* Vol.40, No.2, (1993), pp.96–106.