International Journal for Modern Trends in Science and Technology
Volume 10, Issue 02, pages 429-432.
ISSN: 2455-3778 online
Available online at: http://www.ijmtst.com/vol10issue02.html
DOI: https://doi.org/10.46501/IJMTST1002057

Double Text Data Compression: New LZW Algorithm with Bit Reduction Algorithm

N Srinivasa Rao, M Praveen Kumar, K Sai Prasanth, Mastanaih Naidu Y

Department of Information Technology, Bapatla Engineering College, Bapatla, AP, India.

To Cite this Article
N Srinivasa Rao, M Praveen Kumar, K Sai Prasanth, Mastanaih Naidu Y, Double Text Data Compression: New LZW Algorithm with Bit Reduction Algorithm, International Journal for Modern Trends in Science and Technology, 2024, 10(02), pages. 429-432.https://doi.org/10.46501/IJMTST1002057

Article Info
Received: 28 January 2024; Accepted: 19 February 2024; Published: 25 February 2024.

ABSTRACT

The process of data compression entails the reduction of bits required to represent information, providing benefits such as reducing the usage of valuable resources, like disk space and transmission bandwidth. It is widely integrated into various technologies, such as storage systems, databases, operating systems, and software applications, making it crucial to choose an appropriate compression algorithm. This paper proposes combining two techniques, namely modified LZW with dynamic bit reduction achieve a superior compression ratio. This method helps address the issue of costly resources associated with data, specifically disk storage and bandwidth. Further development of the proposed system architecture could involve combining text and image compression to lessen the strain of transmitting and storing significant files via mail or social media websites such as Facebook or Instagram.

KEYWORDS: Text Data Compression, Decompression, compression ratio, LZW, Decoding, Encoding.

1. INTRODUCTION

The concept of data compression is the process of transforming a file, such as text, audio, or video, into a compressed version, which can be fully recovered without losing any vital information. This process is useful in saving storage space, especially if the original file size is too large. For instance, a 4MB file can be compressed to a smaller size to conserve storage space. Moreover, compressed files are easily exchanged over the internet since they can be uploaded and downloaded much faster. Therefore, it is imperative to be able to reconstitute the original file from the compressed version whenever required. Data compression involves encoding rules that can significantly reduce the number of bits required to store or transmit a file, thereby saving on storage and transmission costs.

In essence, data compression is the technique of encoding data into fewer bits than the original representation, resulting in less storage space and shorter transmission time while communicating over a network. This method is possible because real-world data is highly redundant, meaning it contains lots of repetitive information. Therefore, data compression is defined as a method of reducing data size by using different techniques that can be lossy or lossless. To compress data, a compression program is used to convert it from an easy-to-use format to an optimized version that is more compact. Similarly, an uncompressing program can restore the information to its original form.

Two primary types [8] of data compression exist, each with distinct applications. Lossy data compression [2] is frequently utilized to reduce the size of image files for communication or archival purposes. In contrast, lossless data compression is typically utilized to preserve the integrity of text or binary files during transmission or archival storage. These two classes are clearly delineated: lossy compression and lossless compression.

2. LITERATURE SURVEY

Data compression comes in different forms, one of which is known as lossy compression. In this type of compression, the decompressed data may not be an exact replica of the original data, but it is close enough to serve its intended purpose. Lossy compression should not be used for critical data such as text since the original message cannot be recovered once it has been compressed. However, it is particularly useful for Digitally Sampled Analog Data (DSAD) which mainly comprises of sound, video, graphics, or picture files. Lossy compression algorithms like JPEG, MPEG, and MP3 are widely used on the internet, especially in streaming media and telephony applications. One disadvantage of lossy data compression is the potential for generation loss when files are repeatedly compressed and decompressed, leading to a decrease in text quality. Nonetheless, lossy image compression is frequently used in digital cameras to save space without compromising the quality of pictures.

2.1 Original LZW Data Compression

Lempel-Ziv-Welch (LZW [4]) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. Lempel- Ziv-Weltch (LZW) is one of the powerful existing compression algorithms. It finds in many important applications like win zip, 7zip and etc.

1. Original LZW is a fixed length coding algorithm. Uses 12bit unsigned codes. First 256 codes are the entire ASCII character set.
2. Lateral entries in the LZW dictionary [7] are strings and codes.
3. Every LZW code word is a reference to a string in the dictionary.

LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.

Basic idea [3]

(1) Replaces strings of characters with single integer codes.
(2) A table of string/code pairs is built as the compression algorithm reads the input file.

(3) The table is reconstructed as the decompression algorithm reads.

2.2 Modified LZW [10]: It can facilitate variable length coding while compression.

Algorithm-1

Modified LZW Compression Algorithm [5]:

```
1: DEFINE CODE LENGTH
2. if (STR = get input character) = EOF then
3: while there are still input characters do
4: CHAR = get input character
5: if STR+CHAR is in the String table then
6: STR = STR+CHAR
7: else
8: output code for STR
9: add STR + CHAR into the String table
10: STR = CHAR
11: end if
12: end while
13: Output the code for STR
14: end if
```

Algorithm-2

Modified LZW Decompression Algorithm:

```
1: DEFINE CODE LENGTH
2. Read OC = OLD CODE
```

```
3: if OC is not EOF then
4: output OC
5: CHARACTER = OC
6: while there are still input characters do
7: Read NC = NEW CODE
8: if NC is in not DICTIONARY, then
9: STRING = get translation of OC
10: STRING = STRING + CHARACTER
11: else
12: STRING = get translation of NC
13: end if
14: output STRING
15: CHARACTER = first character in STRING
16: add OC + CHARACTER into the DICTIONARY
17: OC = NC
18: end while
19: Output string for code
20: end if
```

2.3 Bit Reduction Algorithm [2]:

Data compression is always useful for encoding information using lesser number of bits than the original representation it would use. There are many applications where the size of information would be critical. In data communication, the size of data can affect the storage cost. This algorithm was originally implemented for use in a text file like message communication application. The idea in is this program reduces the standard 8-bit encoding to some application using specific 5-bit encoding system and then pack into a byte array. This method will reduce the size of a string considerably when the string is lengthy and the compression ratio is not affected by the content of the string. The Algorithm

1. Compression (Encoding [4]):

Let's assume that we have a input string with 8 characters. If we put this on a byte array, we get a byte array with the size of 8. A single character will need 8 bits if the characters are represented with ASCII values. A set of 8 bits can represent 2^8 different characters. But if we consider the application, a simple text data might be included only around 26 different characters. Therefore, it is need to have 5-bit encoding which can give up to 2^5 different characters to represent. For converting into the new 5-bit encoding, we assign new values to the alphabet characters like | p=1 | q=2 | r=3 | s=4 | t=5 | u=6 | v=7 | w=8 |. If we look more closely at the new byte array, it will look like the following (the values of characters are in binary representation). 00000001 | 00000010 | 00000011 | 00000100 | 00000 101 | 00000110 | 00000111 | 00001000| But we use 8 bytes for storing the 8 characters. In the next step, we will cut three bits from the position of 3rd bit from the left side and extract the 5 least significant bits. The result will be shows as follows: |00001|00010|00011|00100|00101|00110|00111|01000|. Now we have reduced 8 bytes to 5 bytes. The next step shows how these 5 bytes convert to the 8 bytes and we get the original information.

2. Decompression (Decoding):

When an array of bytes is shown, each character should be represented in the binary form. Then all the 1's and 0's should be arranged as their index values and all data split to the sets of five bits. After splitting data, it will be as follows: Code |00001 000|10 00011 0|0100 0010|1 0011000|111 01000| then these sets converted to decimal values represent the characters that we have compressed. Code |00001 = 1(a) 000|10 = 2 (b) 00011 = 3(c) 0|0100 = 4 (d) 0010|1 = 5 (e) 00110 = 6 (f) 00|111 = 7 (g) 01000| = 8 (h). Then the information will be shown in original form as "abcdefgh".

3. DESIGN

In the design we can achieve better compression combining the both techniques one after one at the time compressing the text data. First apply the new approach in LZW algorithm to compress the text data, and second use the data produced by first technique encoded information as input and apply this technique.

1. Text Data Compression Algorithm

Step I: Input the modified LZW result text data to be compressed.

Step II: Find the number of unique words in the input text data and assign the symbols that are not in the input.

Step III: Now find the unique characters from the step II.

Step IV: Find the number of bits required to assign bit code to the characters.
Step V: Assign the numeric code to the unique characters found in the step II according to the number of bits calculated in step IV.
Step VI: Starting from first symbol in the input find the binary code corresponding to that symbols from assigned numerical codes and concatenate them to obtain binary output.
Step VII: Add number of 0's in MSB of Binary output until it is divisible by 8.
Step VIII: Generate the ASCII code for every 8 bits for the binary output obtained in step VI and concatenate them to create input for second phase.
[Step VI is the result of dynamic bit Reduction method in ASCII format]
Step IX: Compressed data is obtained.
2. Text Data Decompression Algorithm
Step I: Input the Final output from compressed phase.
Step II: Calculate the binary code corresponding to the ASCII values of input obtained in Step I.
Step III: Remove the extra bits from the binary output added in the compression phase.
Step IV: Calculate the numeric code for every 8 bits obtained in the Step IV.
Step V: For every numeric value obtained in the step V, find the corresponding symbol to get the final decompressed data.
Step VI: Concatenate the data symbols obtained in the step VI and obtain the final output.
Step VII: Display the final result to the user.

4. CONCLUSION & FUTURE WORK

Modified LZW algorithm with Dynamic bit reduction techniques are presented in this paper. It concludes that we can achieve the goal to reduce the actual file size into a better compressed file. This indicates that combining two techniques have performed better than the existing LZW algorithm and one time compression in terms of compressed file size. Limitations of this work include dictionary overflow with large files and increased searching time.
The suggested future work is reducing the time of compression using multi programming technique on this paper.

Conflict of interest statement

Authors declare that they do not have any conflict of interest.

REFERENCES

[1] Srinivasa Rao Namburi, Praveen Kumar Muvva, A New Approach To Increase Lzw Algorithm Compression Ratio, IJEAST, Vol. 4 Issue 10, pg. 141-144, 2020.

[2] Rajinder Kaur, Er. Monica Goyal, An Algorithm For Lossless Text Data Compression, IJERT, vol. 2 Issue 7, 2013.

[3] David Solomon. Data compression: The complete references book, Pub-SV 3rd Edition, 2004.

[4] Michael Dipperstein. Lempel-Ziv-Welch (LZW) Encoding Discussion, http://michael.dipperstein.com/lzw/.

[5] Simrandeep kaur, V.Sulochana Verma, Design and Implementation of LZW Data Compression Algorithm, International Journal of Information Sciences and Techniques (IJIST) Vol.2, No.4, July 2012.

[6] Evon Abu-Taieh1, Issam AlHadid, A New Lossless Compression Algorithm, Modern Applied Science, Canadian Center of Science and Education, Vol. 12, No. 11, 2018.

[7] Restu Maulunida, Achmad Solichin, Optimization of LZW Compression Algorithm With Modification of Dictionary Formation, Indonesian Journal of Computing and Cybernetics Systems) Vol.12, No.1, January 2018, pp. 73~82.

[8] J.Uthayakumar, T. Vengattaraman, P. Dhavachelva, A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications, Journal of King Saud University - Computer and Information Sciences, 2018.