A New Algorithm for Lossless Data Compression based of Time Clock Mechanism

Chandni Keshri
Department of CSE
Amity University
Greater Noida, India

Bhanu Prakash Lohani
Department of CSE
Amity University
Greater Noida, India

ABSTRACT

Data compression is pivotal for optimizing storage and enhancing transmission efficiency in modern computing systems. This paper introduces a novel lossless data compression algorithm based on a time clock mechanism. The proposed algorithm, termed Time Clock Compression (TCC), leverages a temporal encoding strategy to minimize redundancy by effectively capturing data patterns with high temporal precision. The results indicate that TCC is particularly advantageous for real-time applications and large-scale data environments where rapid processing and efficient storage are critical. This innovation presents a promising data compression, offering a robust solution for various industries reliant on efficient data management.

General Terms

Compression Ratio, Redundancy, Dictionary, Indexing, Entropy.

Keywords

List_Unique, String_ASCII, Matrix_3x, String_Size.

INTRODUCTION

Data compression is a fundamental technology in digital systems, enabling efficient storage and transmission of data. Traditional lossless compression algorithms, such as Huffman Coding and Lempel-Ziv-Welch (LZW), have been widely used for their ability to reduce file size without any loss of information. However, as data volumes continue to grow exponentially, there is a need for more efficient compression techniques that can handle large-scale data and real-time processing requirements.

TYPES OF DATA COMPRESSION

We can categorize data compression in following way.

Lossless Compression

Lossless compression algorithms reduce file size without losing any information, allowing the original data to be perfectly reconstructed. Common techniques include:

- Huffman Coding: A variable-length encoding algorithm that assigns shorter codes to more frequent symbols.
- Lempel-Ziv-Welch (LZW): Utilizes a dictionary-based approach to replace repeated sequences with shorter codes.
- Run-Length Encoding (RLE): Compresses data by encoding consecutive repeated characters as a single character and count.

2.2 Lossy Compression

Lossy compression reduces file size by removing some information, typically imperceptible to human senses. It is widely used for multimedia applications:

- JPEG (Joint Photographic Experts Group): A method for compressing photographic images by reducing redundant data and allowing a trade-off between quality and file size.
- MP3 (MPEG Audio Layer III): A common audio compression format that removes inaudible frequencies and reduces bit rates.
- H.264: A video compression standard that achieves high compression rates through inter-frame prediction and motion compensation.

3. PROPOSED ALGORITHM

The proposed algorithm employs a time clock mechanism to achieve efficient and lossless compression and analysis of character data. The algorithm operates by segmenting the input string into three columns, where the first column represents hours, the second column represents minutes, and the third column represents seconds. In a traditional clock, the timekeeping system is divided into 60 seconds per minute, 60 minutes per hour, and 12 hours per cycle. However, in our specialized clock system, the divisions are extended to 256 seconds per minute, 256 minutes per hour, and 256 hours per cycle. Also a predefined index with values ranging from 0 to 255 is utilized. Each Character frequencies are counted and higher frequencies are replaced with lower index values. Then, each row's numeric value is calculated using a specific mathematical formula. This method ensures efficient compression without any data loss.

3.1 Data Compression Algorithm Design

There are several step involve to compress data, Those step are following:

Step 1: Input Handling and Padding

a) Input Reading: Retrieve and read the binary input string from the file.
b) Padding: Determine the length of the input binary string. If the total number of characters is not divisible by 24 (i.e., if there is a remainder), calculate the number of additional characters required to make the length divisible by 24. Append '0' at left hand side of the binary string until the required length is achieved.

Step 2: Data Analysis and Matrix Creation

a) Segmentation: Segment the input binary string into blocks of 8 bits each. For each 8-bit segment, convert the binary value to its corresponding ASCII character using the ASCII table.
b) Matrix Creation: Create a matrix structure with 3 columns and a number of rows denoted as ‘x’. The value of ‘x’ is determined by the length of the input string, specifically by dividing the total number of ASCII characters by 3. Begin filling the matrix by placing the ASCII values derived from the input string into the matrix, starting with the first column of the first row, moving horizontally to the second

and third columns. Once the first row is filled, proceed to the next row, continuing to place the ASCII values sequentially from left to right, until all values from the input string are placed into the matrix.

Step 3: Unique Character List
Generate a Table of unique characters by systematically analyzing the ASCII string. For each unique character, determine the column in which it resides within the constructed matrix by starting with Column A. If the character is found in Column A, cease further checks and proceed to the next unique character, repeating the process. If the character is not found in Column A, proceed to check Column B. If found in Column B, stop further checks and move to the next unique character, starting again from Column A. If not found in Column B, check Column C. Record the column location of each character for subsequent processing.

Step 4: Character Indexing & Replacing

a) Frequency Analysis: Calculate the frequency of each character and add this information to Unique character Table. Begin by including the frequencies from column A, then proceed with those from column B, and finalize with the frequencies from column C.
b) Index Assignment : Utilize a predefined index range from 0 to 255. Assign the character with the highest frequency to the lowest index value. This assignment should be performed sequentially based on the frequencies obtained, starting with the characters from column A, then those from column B, and finally from column C.
c) Matrix Substitution: Replace each character in the previously constructed 3x matrix with its corresponding index value as determined in the previous step. This involves mapping each character in the matrix to the index value it was assigned.

Step 5: Binary Encoding and Compression Simulation

a) Calculate Numeric value of each row : Compute the unique value for each row in the matrix using the formula: (col_a×255^ 2) + (col_b×255^ 1) + (col_c×255^ 0). In this formula, col_a, col_b, and col_c represent the values in each column of the row. The result of this calculation will provide a unique numeric value for each row, based on the weighted sum of the column values.
b) Convert Row Values to Binary: Sequentially convert each calculated row value into binary format according to the following conditions:
   1) If the row value is less than 65,535, convert it to binary and ensure that the resulting binary string has exactly 16 bits. If the binary representation is shorter than 16 bits, pad it with leading zeros. Prepend '00' to this 16-bit binary string.
   2) If the row value is less than 524,287, convert it to binary and ensure that the resulting binary string has exactly 19 bits. If the binary representation is shorter than 19 bits, pad it with leading zeros. Prepend '01' to this 19-bit binary string.
   3) If the row value is less than 2,097,151, convert it to binary and ensure that the resulting binary string has exactly 21 bits. If the binary representation is shorter than 21 bits, pad it with leading zeros. Prepend '10' to this 21-bit binary string.
   4) If the row value is greater than 2,097,151, convert it to binary and pad the binary string to a total of 24 bits. Prepend '11' to this 24-bit binary string.

Step 6: Compress Data output.

The compressed data string is generated by sequentially reading each row of the binary strings generated. This process starts from the first row and continues through to the last row of the data set.

3.2 Data Decompression Algorithm Design

Step-1: Initial Bit Analysis
Begin by reading the first 2 bits of the compressed data string. This initial segment will determine the length of the subsequent data segment that needs to be processed.

a) Determine Segment Length: Based on the value of the first 2 bits, select the appropriate length for the next segment of the binary data:
   1) If the first 2 bits are "00", the following 16 bits should be extracted.
   2) If the first 2 bits are "01", extract the next 19 bits.
   3) If the first 2 bits are "10", extract the next 21 bits.
   4) If the first 2 bits are "11", extract the next 24 bits.

b) Extract and Convert Binary Segment:
   1) Extract the designated number of bits from the compressed data string, as determined in the previous step.
   2) Convert this extracted binary segment into its corresponding numeric value.

Example:
For instance, if the compressed input string is "01100010011010110100", follow these steps:

1) The first 2 bits are "01", which indicates that the following segment length is 19 bits.
2) Extract the next 19 bits: "1100010011010110100".
3) Convert this 19-bit binary segment into a numeric value. In this example, the binary segment "1100010011010110100" converts to the numeric value 403,124.

Step-2: We obtain the original data by substituting the generated value (403124) into the equation: (a·255·255) + (b·255) + c = 403124.

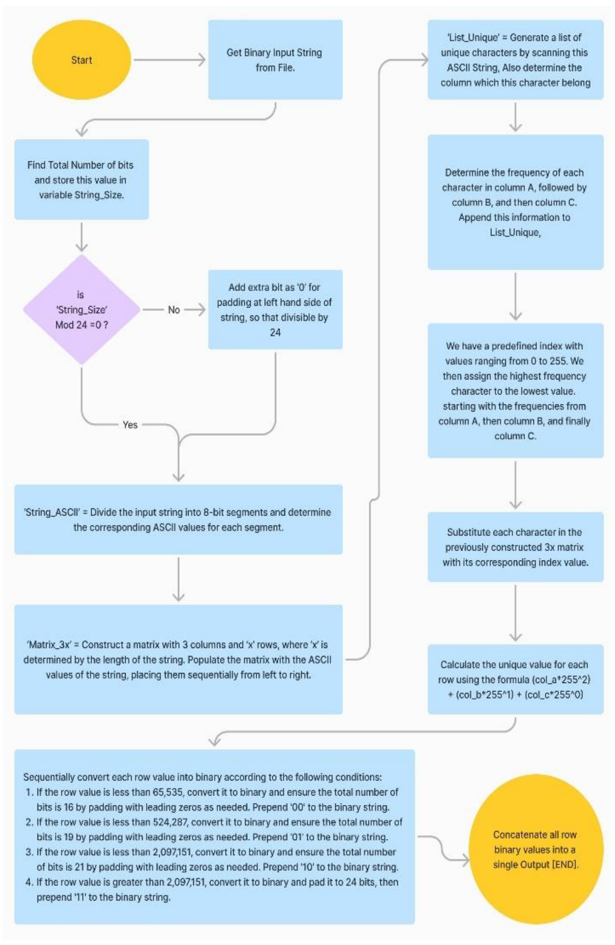
1) After solving this equation, we find that a=6, b=50, and c=224. These are index value.
2) Step-3: We use these index values (6, 50, and 224) to find their corresponding values to retrieve the original string.

4. TIME COMPLEXITY

This algorithm operates with a time complexity of O(n)+ C, where n is the size of the input data. The initialization and divide data into 3 column steps involve linear scans of the data, while the temporal encoding and compression steps also scale linearly with the input size.

5. FLOWCHART

The flowchart below illustrates the operational steps of the This Data Compression algorithm. This visual representation outlines how leverages a clock mechanism to calculate numeric number of each column. The flowchart details the initialization, data analysis, divide data into 3 columns, compression phases, demonstrating the algorithm's efficient handling of input data while maintaining lossless compression integrity.


Start
Get Binary Input String from File.
Find Total Number of bits and store this value in variable String_Size.
is 'String_Size' Mod 24 =0 ?
No
Add extra bit as '0' for padding at left hand side of string, so that divisible by 24
Yes
'String_ASCII' = Divide the input string into 8-bit segments and determine the corresponding ASCII values for each segment.
'Matrix_3x' = Construct a matrix with 3 columns and 'x' rows, where 'x' is determined by the length of the string. Populate the matrix with the ASCII values of the string, placing them sequentially from left to right.
'List_Unique' = Generate a list of unique characters by scanning this ASCII String, Also determine the column which this character belong
Determine the frequency of each character in column A, followed by column B, and then column C. Append this information to List_Unique,
We have a predefined index with values ranging from 0 to 255. We then assign the highest frequency character to the lowest value. starting with the frequencies from column A, then column B, and finally column C.
Substitute each character in the previously constructed 3x matrix with its corresponding index value.
Calculate the unique value for each row using the formula (col_a*255^2) + (col_b*255^1) + (col_c*255^0)
Sequentially convert each row value into binary according to the following conditions:
1. If the row value is less than 65,535, convert it to binary and ensure the total number of bits is 16 by padding with leading zeros as needed. Prepend '00' to the binary string.
2. If the row value is less than 524,287, convert it to binary and ensure the total number of bits is 19 by padding with leading zeros as needed. Prepend '01' to the binary string.
3. If the row value is less than 2,097,151, convert it to binary and ensure the total number of bits is 21 by padding with leading zeros as needed. Prepend '10' to the binary string.
4. If the row value is greater than 2,097,151, convert it to binary and pad it to 24 bits, then prepend '11' to the binary string.
Concatenate all row binary values into a single Output [END].

Flowchart 1: Proposed Data Compress Algorithm

6. APPLICATIONS

There are several application of this algorithm as following:

a) **Real-Time Data Compression:** TCC's ability to quickly adapt to changing data patterns makes it ideal for real-time applications, such as live data streams and online gaming.

b) **Large-Scale Data Storage:** TCC's efficient compression ratios reduce storage requirements, benefiting large-scale data environments like data centers and cloud storage services.

7. CONCLUSION

This algorithm offers a lossless data compression method by leveraging a time clock mechanism to enhance both compression efficiency and processing speed. While it is particularly effective for text data compression, it is versatile and can also be applied to compress all types of data. The experimental results demonstrate data compression result, especially in real-time and large-scale data applications. Future work will focus on further optimizing the algorithm and exploring its potential in various industrial applications.

8. REFERENCES

[1] Sayood, K. (2012). Introduction to Data Compression. Elsevier.

[2] Salomon, D., & Motta, G. (2010). Handbook of Data Compression. Springer.

[3] Pennebaker, W. B., & Mitchell, J. L. (1993). JPEG Still Image Data Compression Standard. Springer.

[4] Ziv, J., & Lempel, A. (1977). A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory, 23(3), 337-343.

[5] Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE, 40(9), 1098-1101.

[6] M. A. Bassiouni, "Data Compression in Scientific and Statistical Databases", IEEE Transactions on Software Engineering, vol. SE-11, no. 10, pp. 1047-1058, 1985.

[7] R. Bayer and E. McCreight, "Organization and maintenance of large ordered indexes", Acta Informat, vol. 1, pp. 173-189, 1972.

[8] J. Goldstein, R. Ramakrishnan and U. Shaft, Compressing Relations and Indexes, December 1997.

[9] V. Kesri, " A New Algorithm to Provide all Solutions of SSP Problem" International Journal of Computer Applications, vol. 178, number 5, p. 20–25, 2017.

[10] Christopher H. Messom, Gourab Sen Gupta and Serge N. Demidenko “Hough Transform Run Length Encoding for Real Time Image Processing” IEEE transactions on instrumentation and measurement, vol. 56, no. 3, June 2007.

[11] Nikolay Manchev “Parallel algorithm for Run Length Encoding”, Proceedings of third international conference on information theory, 2006.

[12] J. Trein, A.Th.Schwarzbacher, B. Hoppe and K.-H. Noffz “A Hardware Implementation of a Run Length Encoding Compression Algorithm with Parallel Inputs”, ISSC 2008, Galway, June 18-19.

9. APPENDIX-1

<u>STEP-1: Input Reading</u>

Input String:
0100000100100000011000110110000101110100
00100000011100100110000101101110001000000110000101
11011101100001011110010010110000100000011000110110
10000110000101110011011010010110111001100111001000
000110000100100000001110010011000010111010000101110

Padding

String_Size= 240
[Find total number of bits in Input string]
240(mod 24) = 0
[Its divisible by 24 so that padding not required]

<u>STEP-2: Segmentation</u>

To perform segmentation, we divide into 8 bits blocks and find their corresponding ASCII value. It look like following:

Table 1. ASCII Value of segmented string

Binary	ASCII Number	ASCII Value
01000001	65	A
00100000	32	(Space)
01100011	99	c
01100001	97	a
01110100	116	t

00100000	32	(Space)
01110010	114	r
01100001	97	a
01101110	110	n
00100000	32	(Space)
01100001	97	a
01110111	119	w
01100001	97	a
01111001	121	y
00101100	44	,
00100000	32	(Space)
01100011	99	c
01101000	104	h
01100001	97	a
01110011	115	s
01101001	105	i
01101110	110	n
01100111	103	g
00100000	32	(Space)
01100001	97	a
00100000	32	(Space)
01110010	114	r
01100001	97	a
01110100	116	t
00101110	46	.

Table 2. Matrix Creation

Column A	Column B	Column C
A	(Space)	c
a	t	(Space)
r	a	n
(Space)	a	w
a	y	,
(Space)	c	h
a	s	i
n	g	(Space)
a	(Space)	r
a	t	.

<u>STEP-3: Find Unique Character</u>

Table 3. Scan unique character of Input String

S. No	Unique Character	Belonging Column	Frequency
1	A (uppercase)	Column A	
2	c	Column B	
3	a (lowercase)	Column A	
4	t	Column B	
5	r	Column A	
6	n	Column A	
7	w	Column C	
8	y	Column B	
9	, (comma)	Column C	
10	(space)	Column A	
11	h	Column C	
12	s	Column B	
13	i	Column C	
14	g	Column B	
15	.	Column C	

<u>STEP-4 Frequency Analysis</u>

Table 4. Find frequency of each unique character

S. No	Unique Character	Belonging Column	Frequency
1	A (uppercase)	Column A	1
2	c	Column B	2
3	a (lowercase)	Column A	6
4	t	Column B	3
5	r	Column A	2
6	n	Column A	2
7	w	Column C	1
8	y	Column B	1
9	, (comma)	Column C	1
10	(space)	Column A	6
11	H	Column C	1
12	S	Column B	1
13	I	Column C	1
14	G	Column B	1
15	.	Column C	1

Table 5. Index Assignment

Index Value	Character	Belonging Column	Frequency
0	a (lowercase)	Column A	6
1	(space)	Column A	6
2	r	Column A	2
3	n	Column A	2
4	A (uppercase)	Column A	1
5	t	Column B	3
6	c	Column B	2
7	y	Column B	1
8	s	Column B	1

9	g	Column B	1
10	w	Column C	1
11	, (comma)	Column C	1
12	h	Column C	1
13	i	Column C	1
14	.	Column C	1
15			
.			
Upto 255			

Table 6. Matrix Substitution

Column A	**Column B**	**Column C**
4	1	6
0	5	1
2	0	3
1	0	10
0	7	11
1	6	12
0	8	13
3	9	1
0	1	2
0	5	14

STEP-5: Calculate Numeric value of each row

Table 7. Numeric value of each row

Column A	Column B	Column C	Row Value
4	1	6	260361
0	5	1	1276
2	0	3	130053
1	0	10	65035
0	7	11	1796
1	6	12	66567
0	8	13	2053
3	9	1	197371
0	1	2	257
0	5	14	1289

Table 8. Convert Row numeric values to Binary

Row Value	**Check Value**	**LHS bits Padding**	**Final String**
260361	<524287	01	0101111111100100001001
1276	<65535	00	000000010011111100
130053	<524287	01	0100111111110000000101
65035	<65535	00	001111111000001011
1796	<65535	00	000000011100000100
66567	<524287	01	0100100000010000000111
2053	<65535	00	000000100000000101
197371	<524287	01	0101100000001011111011
257	<65535	00	000000000100000001
1289	<65535	00	000000010100001001

STEP-6: Compress Data Output:
0101111111100100001001000000001001111111000100111111
0000000101001111111000001011000000001110000001000100
1000001000000001110000000100000000101010110000001011
111011000000000010000000010000000010100001001

Total Number of Bits is = 192