Data compression is a very useful technique that helps in reducing the size of text data and storing the same amount of data in relatively fewer bits resulting in reducing the required data storage space. Text data compression techniques can be categorized as Lossy and Lossless. In this paper, we propose an Improved Dynamic Bit Reduction Algorithm for compression and decompression of text data based on lossless data compression approach. Improved Dynamic bit reduction algorithm identifies the no. of unique characters in the input string and performs the compression process by converting the ASCII characters in to binary code dynamically based upon the occurrence of unique symbols in the input string. Huffman coding algorithm is used to optimize the performance of dynamic bit reduction. The performance of the proposed algorithm is measured and compared with the existing systems using parameters- Compression Ratio and Saving Percentage.

Keywords— Compression ratio, Dynamic Bit reduction, Huffman coding, Lossless compression, Saving Percentage, Text Data Compression

I. INTRODUCTION

Data Compression is the process of encoding data to fewer bits so that it takes less storage space or less transmission time that it would if it were not compressed [1].

![Diagrammatic view of Compression](image)

Compression is possible because most of the real world data is very redundant. Data Compression is a technique that reduces the size of data by removing unnecessary information and duplicity. Data compression is an art of reducing the number of bits needed to store or transmit data.

Mainly, two types of data compression techniques are there: Lossy and Lossless data compression [1].

A lossy data compression method is one where compressing data and then decompressing it retrieves data that may not be exactly same as the original, but is "close enough" to be useful for specific purpose. The examples of frequent use of Lossy data compression are on the Internet and especially in the streaming media and telephony applications. Most of the lossy data compression techniques suffer from generation loss which means decreasing the quality of text because of repeatedly compressing and decompressing the file. Lossy image compression can be used in digital cameras to increase storage capacities with minimal degradation of picture quality.

Lossless data compression is a technique that allows the use of data compression algorithms to compress the data and also allows the exact original data to be reconstructed from the compressed data. This is in contrary to the lossy data compression in which the exact original data cannot be reconstructed from the compressed data. The popular ZIP file format that is being used for the compression of data files is also an application of lossless data compression approach. Lossless compression is used when it is important that the original data and the decompressed data be identical. Lossless text data compression algorithms usually exploit statistical redundancy in such a way so as to represent the sender's data more concisely without any error or any sort of loss of important information contained within the text input data. Since most of the real-world data has statistical redundancy, therefore lossless data compression is possible. For instance, In English text, the letter ‘a’ is much more common than the letter ‘z’, and the probability that the letter ‘t’ will be followed by
the letter ‘z’ is very small. So this type of redundancy can be removed using lossless compression. Lossless compression methods may be categorized according to the type of data they are designed to compress. Compression algorithms are basically used for the compression of text, images and sound. Most lossless compression programs use two different kinds of algorithms: one which generates a statistical model for the input data and another which maps the input data to bit strings using this model in such a way that frequently encountered data will produce shorter output than improbable (less frequent) data. The advantage of lossless methods over lossy methods is that Lossless compression results are in a closer representation of the original input data. The performance of the algorithms can be measured using the parameters such as Compression Ratio and Saving Percentage.

**Compression Ratio:** Compression ratio is defined as the ratio of size of the compressed file to the size of the source file [2].

\[
\text{Compression ratio} = \frac{C_2}{C_1} \times 100\%
\]

**Saving Percentage:** Saving Percentage calculates the shrinkage of the source file as a percentage.

\[
\text{Saving percentage} = \left(1 - \frac{C_2}{C_1}\right) \times 100\%
\]

C1 = Size before compression
C2 = Size after compression

## II. EXISTING WORK

### A. Bit Reduction algorithm:

The main idea behind this program is to reduce the standard 7-bit encoding to some application specific 5-bit encoding system and then pack into a byte array. This algorithm was originally implemented to use in a text file like message communication application [3].

**Bit Reduction Algorithm in steps:**

- Select the frequently occurring characters from the text file which are to be encoded and obtain their corresponding ASCII code.
- Obtain the corresponding binary code of these ASCII key codes for each character.
- Then put these binary numbers into an array of byte (8 bit array).
- Remove extra bits from binary no like extra 3 bits from the front.
- Then rearrange these into array of byte and maintain the array.
- Final text will be encoded and as well as compression will be achieved.
- Now decompression will be achieved in reverse order at the client-side.

### B. Huffman Coding:

Huffman coding deals with data compression of ASCII characters. It follows top down approach means the binary tree is built from the top down to generate an optimal result. In Huffman Coding the characters in a data file are converted to binary code and the most common characters in the file have the shortest binary codes, and the characters which are least common have the longest binary code [4]. The algorithm to generate Huffman code is:

1. **Parse the input and count the occurrence of each symbol.**
2. **Determine the probability of occurrence of each symbol using the symbol count.**
3. **Sort the symbols according to their probability of occurrence, with the most probable first.**
4. **Then generate leaf nodes for each symbol and add them to a queue.**
5. **Take two least frequent characters and then logically group them together to obtain their combined frequency that leads to the construction of a binary tree structure.**
6. **Repeat step 5 until all elements are reached and there remains only one parent for all nodes which is known as root.**
7. **Then label the edges from each parent to its left child with the digit 0 and the edge to right child with 1. Tracing down the tree yields to “Huffman codes” in which shortest codes are assigned to the character with greater frequency.**

## III. PROPOSED SYSTEM

Proposed System works in two phases to compress the text data. In the first phase data is compressed with the help of dynamic bit reduction technique and in second phase Huffman coding is used to compress the data further to produce the final output. In the similar way method of decompression works in reverse order. Compressed data is first decompressed by Huffman Decoder and then by dynamic bit reduction decoder. Following are the steps to compress the data with the help of our proposed system:

**Steps for Data Compression:**

Step I: Input the text data to be compressed.

Step II: Find the number of unique symbols in the input text data.

Step III: Assign the numeric code to the unique symbols found in the step II.

Step IV: 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 V: Add number of 0’s in MSB of Binary output until it is divisible by 8.
Step VI: Generate the ASCII code for every 8 bits for the binary output obtained in step V and concatenate them to create input for second phase.
[Step VI is the result of dynamic bit Reduction Method]
Step VII: Give the output generated by Step VI to Huffman tree to further compress the data and obtain the result.
Step VIII: Display the final result obtained in step VII.
[Output from step VIII is final compressed output]

Steps for Decompression
Step I: Input the Final output from compressed phase.
Step II: Assign this input to the Huffman decoder to decompress the data compressed by Huffman tree in ASCII format.
Step III: Calculate the binary code corresponding to the ASCII values obtained in Step II.
Step IV: Remove the extra bits from the binary output added in the compression phase.
Step V: Calculate the numeric code for every 8 bits obtained in the Step IV.
Step VI: For every numeric value obtained in the step V, find the corresponding symbol to get the final decompressed data.
Step VII: Concatenate the data symbols obtained in the step VI and obtain the final output.
Step VIII: Display the final result to the user.

IV. RESULTS AND DISCUSSION
This section involves the implementation of existing and the proposed system and analysing their performance in terms of Compression Ratio.

A. Bit Reduction Output
Implementation: This algorithm has been implemented using C# programming language and VS 2008 as an Integrated Development Environment.

![Snapshot](image)

Fig. 2 Output of Bit reduction algorithm

<table>
<thead>
<tr>
<th>Input text size (in bytes)</th>
<th>Compressed Output (in bytes)</th>
<th>Saving percentage</th>
<th>Compression ratio (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>66</td>
<td>24.1</td>
<td>75.8</td>
</tr>
<tr>
<td>117</td>
<td>88</td>
<td>24.7</td>
<td>75.2</td>
</tr>
<tr>
<td>176</td>
<td>132</td>
<td>25</td>
<td>75</td>
</tr>
<tr>
<td>352</td>
<td>264</td>
<td>25</td>
<td>75</td>
</tr>
<tr>
<td>536</td>
<td>402</td>
<td>25</td>
<td>75</td>
</tr>
</tbody>
</table>

Table I shows the various experiments conducted by the authors to determine the compression ratio achieved by bit reduction algorithm for text data of varying lengths. The compression ratio of bit reduction algorithm lies around 75% which means that it saves around 25% of the storage space in memory.
B. Huffman coding Output
Implementation: This algorithm has also been implemented using C# programming language and VS 2008 as an Integrated Development Environment.

Table II shows the various experiments conducted by the authors to determine the compression ratio achieved by Huffman coding algorithm for text data of varying lengths.

C. Proposed System Output
Implementation: This algorithm has also been implemented using C# programming language and VS 2008 as an Integrated Development Environment.
Table III shows the various experiments conducted by the authors to determine the compression ratio achieved by the final proposed system. The compression ratio of the proposed system lies between 29-53% which means that it saves around 47-71% of the storage space in memory which is better than the existing techniques.

D. Comparison Tables and Graphs
The following table and graph represents the comparison of Compression ratios of the existing techniques and the proposed system.

<table>
<thead>
<tr>
<th>Input text size (in bytes)</th>
<th>Output of proposed System (in bytes)</th>
<th>Saving Percentage</th>
<th>Compression Ratio of proposed system (In %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>26</td>
<td>70.1</td>
<td>29.8</td>
</tr>
<tr>
<td>117</td>
<td>36</td>
<td>69.2</td>
<td>30.7</td>
</tr>
<tr>
<td>176</td>
<td>87</td>
<td>50.5</td>
<td>49.4</td>
</tr>
<tr>
<td>352</td>
<td>174</td>
<td>50.5</td>
<td>49.4</td>
</tr>
<tr>
<td>536</td>
<td>286</td>
<td>46.6</td>
<td>53.3</td>
</tr>
</tbody>
</table>

From Table IV, it is clear that the compression ratio achieved by the proposed system is lesser as compared to the existing techniques which means it results in more savings of the storage space.

The above graph (Fig. 5) is made on the basis of Table IV which shows the comparison of compression ratios of the existing systems and the proposed system. In the graph, the horizontal axis represents the length of input string in bytes and vertical axis represents the Compression Ratio in percentage.

The following table and graph represents the comparison of Saving Percentage of the existing techniques and the proposed system:
V. CONCLUSION

In this paper, an Improved Dynamic Bit Reduction Algorithm to compress and decompress the text data has been introduced. We also present the results obtained by the proposed system and compare these results with the existing data compression techniques. The limitation of the existing bit reduction algorithm is that it does not provide good compression results and loses output if special characters are present in the input string. These limitations have been overcome by the proposed system. Proposed system shows very good compression results in terms of compression ratio and saving percentage and proved better than the existing techniques for these parameters. Improved dynamic bit reduction algorithm works only with text data which can also be tested to compress the multi lingual data for future work.

ACKNOWLEDGEMENT

I would like to thank Dr. Paramjeet Singh, Assistant Professor (CSE), Giani Zail Singh Punjab Technical University Campus, Bathinda and Dr. Shaveta Rani for their valuable suggestions, constant support and guidance. I would also like to express my gratitude towards the Computer Science Department of GZS PTU Campus, Bathinda, India for providing the resources to proceed with the research work.

REFERENCES


From Table V, it is clear that the saving percentage of proposed system is higher as compared to the existing techniques.

<table>
<thead>
<tr>
<th>Input text size (in bytes)</th>
<th>Saving Percentage of Bit Reduction</th>
<th>Saving Percentage of Huffman Coding</th>
<th>Saving Percentage of Proposed System</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>24.1</td>
<td>57.4</td>
<td>70.1</td>
</tr>
<tr>
<td>117</td>
<td>24.7</td>
<td>57.2</td>
<td>69.2</td>
</tr>
<tr>
<td>176</td>
<td>25</td>
<td>42</td>
<td>50.5</td>
</tr>
<tr>
<td>352</td>
<td>25</td>
<td>42</td>
<td>50.5</td>
</tr>
<tr>
<td>536</td>
<td>25</td>
<td>42.1</td>
<td>46.6</td>
</tr>
</tbody>
</table>


