# Modern Lossless Compression Techniques: Review, Comparison and Analysis

Apoorv Gupta[1], Aman Bansal[1], Vidhi Khanduja[2]
1 Department of Information Technology, Netaji Subhas Institute of Technology, Delhi, India
apoorv2711@gmail.com, bansalaman2905@gmail.com
2 Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi, India
vidhikhanduja9@gmail.com

***Abstract*—With the world drifting more and more towards the social network, the size and amount of data shared over the internet is increasing day by day.Since the network bandwidth is always limited, we require efficient compression algorithms to facilitate fast and efficient sharing of data over the network. In this paper, we discuss algorithms of widely used traditional and modern compression techniques. In an effort to find the optimum compression algorithm, we compare commonly used modern compression algorithms: Deflate, Bzip2, LZMA, PPMd and PPMonstr by analyzing their performance on Silesia corpus. The corpus comprises of files of varied type and sizes, which accurately simulates the vast diversity of files shared over the internet. We compare these algorithms on the basis of their compression ratio, compression speed and decompression speed. After observing the simulated results, we found that PPMonstr provides the best compression ratio. Deflate is the fastest algorithm in terms of compression and decompression speed, but provides low compression ratio. Bzip2 and PPMd provide moderate compression speed and good compression ratio and hence are well suited for applications dependent on both compression ratio and speed. Further, experimental results indicate that LZMA is an ideal algorithm for applications requiring high compression ratio along with fast decompression speed as it provides moderate decompression speed coupled with high compression ratio.**

***Keywords—compression; lossless compression; network bandwidth; compression ratio; compression speed; decompression speed; Silesia corpus***

## I. INTRODUCTION

As the world moves forward into the digital age, our dependence on electronic media is increasing exponentially each year. As of every minute in 2015, 300 hours of videos are uploaded on YouTube, more than 250,000 photos are uploaded on Snapchat, 110,000 calls are made on Skype, which is just a fraction of the total data produced and transmitted [1]. Data compression techniques are used to reduce the size of the data which enables its storage and transmission over a limited bandwidth channel. Fig.1 shows basic components of a data compression system. The input data is processed by a Data Compressor, which usually iterates over the data twice. In the first iteration, the compression algorithm tries to gain knowledge about the data which in turn is used for efficient compression in the second iteration. The compressed data along with the additional data used for efficient compression is then stored or transmitted through a network to the receiver. The receiver then decompresses the data using a decompression algorithm. Though data compression can reduce the bandwidth usage of a network and the storage space, it requires additional computational resources for compression and decompression of data, and this additional processing may be harmful to some applications such as embedded devices which have very limited processing capability[2]. Hence, a compression algorithm needs to find a balance between the compression ratio and compression/decompression time.

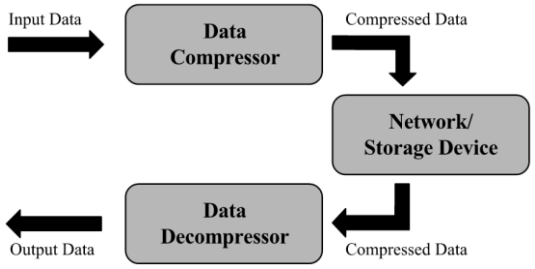


Fig 1: Components of Data Compression System

Compression techniques can be divided into two types, (a)lossless and (b)lossy [2]. As the name suggests, lossless compression techniques do not lead to any loss of data or information. The compression is done by representing the file using less number of bits, without any loss of information. This is done with the help of various mathematical and statistical tools such as entropy coding, which are also used to convert back the compressed file into the original uncompressed file. Lossless compression techniques are used to compress files where it is necessary to retain all the information, as in the case of executable files. Popular compression techniques such as LZ77 [3] and Huffman Coding [4] are examples of lossless compression techniques. On the other hand, lossy compression techniques remove the unnecessary bits, reducing the size of the file. The compressed file cannot be converted back to the original file as some part of the information is lost in the compression. The file recovered after decompression is an approximation of the original file, which is reconstructed on basis of the compression program's understanding. Since a lot of unnecessary information is removed, compression ratio of lossy compression techniques is generally more than lossless compression techniques. Lossy techniques are widely used in

978-1-5090-3239-6/17/$31.00©2017 IEEE

image and video compression, where a group of pixels can be approximated into a single value using transform encoding or differential encoding. Compression techniques such as JPEG [5] and MP3 [6] are examples of lossy compression techniques.In this work, we focus on comparing various widely used modern lossless compression algorithms. The compression algorithms are compared with respect to their compression ratio and their compression and decompression speed.

The rest of the paper is organised as follows: Section II gives an overview of the related work. In section III widely used traditional and modern compression algorithms are discussed. Section IV compares the algorithms on the basis of their performance on Silesia Corpus.Finally, section V concludes the paper with a summary and suggestions for future work.

## II. Prior work

Humans have been finding ways to compress data since the 19th century. With the invention of the Electrical Telegraph System in 1836, the need to compress data was felt for the first time. Morse Code was developed keeping in mind the frequency of alphabets used in English language and so the most commonly used alphabets were assigned shortercodes [7]. Modern compression techniques started with the development of Information Coding techniques in 1940s. Shannon-Fano coding was developed in 1949 based on the work of Claude Shannon and Robert Fano [8], [9]. The technique assigned code words on the basis of the probability of the blocks. The codes generated by Shannon-Fano coding weren't always optimal and hence in 1952, David A. Huffman developed the Huffman Algorithm which always produced optimal codes [4]. By the 1970s, with the advent of the internet, data files started being stored online. Abraham Lempel and Jacob Ziv developed the LZ77 and LZ78, lossless data compression algorithm in 1978 and 1979 respectively which was very efficient in transferring data over a limited bandwidth channel. Both the algorithms maintain a dictionary of the data encountered, and replace it with a reference to the dictionary [3][10]. The LZ algorithm was further improved by Terry Welch in 1984 and the popular LZW algorithm came into existence. The algorithm achieved very high level of compression and was easy to implement [11]. The recent development in data compression is more focussed on compressing Digital images and videos. The development of compression algorithms such as JPEG [5] and MP3 [6] have led to substantial decrease in the size of image and audio files respectively.

With so many compression algorithms, there has been a need to find the most efficient amongst them. Compression algorithms can be compared on the basis of their compression ratio [12] or additionally on the basis of their compression and decompression time [13], [14]. Most of the review papers on compression [12]-[14] focus primarily on traditional compression algorithms. Reference [12] compared RLE, Huffman coding, Arithmetic coding, LZ77 and LZW. It further suggested the optimal technique for different file types after considering the compression ratio. Reference [15] compared many statistical and dictionary based compression techniques such as Arithmetic coding, Huffman coding, RLE, LZ77, LZW, LZSS, LZH etc. and suggested the best technique for practical applications. Though the study of these traditional compression algorithm is important, the need of the hour is to focus on the analysis and comparison of newer and more efficient compression algorithms which are widely used in popular compression software such as 7zip [16], Gzip [17] etc.

In this work, we briefly study the working of some modern compression algorithms such as PPM [18], LZMA [19]-[20], Bzip2 [21] and Deflate [22]. This is followed by their comparison on the basis oftheir compression ratio, compression speed and decompression speed.

## III. Methodology

In this section we briefly discuss various Lossless Compression Techniques used in our analysis. Almost all the newer compression algorithms use some combination of traditional compression algorithms. Hence, it is essential to discuss the traditional algorithms first, before proceeding to more complex compression algorithms.

### A. Traditional Compression Algorithms

*1) Run Length Encoding (RLE)*: It is a very basic method of lossless data compression. The data is read and repetitive symbols are replaced by an escape character followed by a symbol (called 'run') and the binary count [12]. Since each repetition requires three characters for representation (escape, symbol and count), a repetition of less than 4 characters is not considered. Fig.2 describes the alogrithm in detail.

*2) Huffman Encoding:* It is a more efficient method of compression as compared to RLE. Huffman coding is based on the frequency of the characters and hence, the highest frequency character get the shortest binary code whilethe least frequent get the longest [4]. Fig. 3 describes the alogrithm in detail.

*3) LZ-77 Encoding:* In 1977, Abraham Lempel and Jacob Ziv invented a dictionary based compression algorithm, LZ77 [3], [10] which works by exploiting the repetition of words and phrases in a file. A concept of sliding window is used to make references to previously read data [20]. If a word or phrase is repeated, a reference pointer is set to the previous occurrence

| Algorithm 1:**Run_Length_Encoding** |
|---|
| 1: The algorithm looks for repetition of symbols.<br>2: The repetitive characters are replaced by a escape character followed by the symbol and the binary count. |

Fig 2: Algorithm of Run Length Encoding

| Algorithm 2:**Huffman_Encoding** |
|---|
| 1: The Algorithm goes through the data and creates a frequency table of the symbols.<br>2: The two least frequent symbols of the table are removed from the table. They are grouped together and their frequencies are added to form a single node. This node is then added back to the frequency table.<br>3: Step 2 is repeated till a single node is left in the table.<br>4: A tree like structure is formed (called Huffman tree). Traversal of the tree generates the Huffman codes for each symbol. |

Fig 3: Algorithm of Huffman Encoding

of that phrase. Fig. 4 describes the alogrithm in detail.

### B. *Modern Compression Algorithms*

*1) Deflate:* Deflate is the evolutionary algorithm developed by Phillip W. Katz [23]. It uses both LZ77 and Huffman coding for compressing the data [22]. It was designed originally for *.zip* file format but now-a-days, various versions of original Deflate are used for different file formats or softwares like GZIP, 7-zip etc. Fig. 5 describes the alogrithm in detail.

*2) Lempel Ziv Markov Chain Algorithm (LZMA):*In 1998, a variant of LZ77 algorithm, Lempel Ziv Markov Chain Algorithm was developed. It was first used in 7zip application tool as a compression algorithm [19], [20]. LZMA provides high compression ratio when compressing unknown byte stream. The LZMA algorithm uses sliding window compression technique of LZ77 algorithm [24] along with Delta filter and Range Encoder. Fig. 6 describes the alogrithm in detail.

*3) Bzip2:* Bzip2 is a file compression software program published by Julian Seward in 1996 [21]. It is a multilayer advanced compression program that provides excellentcompression results on the expense of space and processing time. Bzip2 divides the data in blocks of size between 100KB to 900 KB and then compresses each block individually using Burrow Wheeler Technique [25]. Fig. 7 describes the algorithm in detail.

*4) Prediction By Partial Matching(PPM) Compression:* Cleary and Witten developed the "Prediction by Partial Matching" data compression algorithm in 1984 [18]. The algorithm provided very good compression ratios and couldrepresent English text in as low as 2.2 bits/character [18]. PPMpredicts the next symbol according to the *n* preceding symbols, which is basically a markov model of $n^{th}$ order. Fig. 8 describes the algorithm in detail.

| Algorithm 3:**LZ-77_Encoding** |
|---|
| 1: The input file is traversed from the beginning, reading a single character at a time.<br>2: A buffer, termed as sliding window is maintained, which stores the occurrence of previous symbols.<br>3: When a repeated symbol is found, it is stored as a reference pointer tothe location of the previous occurrence of the symbol in the sliding window and the number of symbols that are matched. |

Fig 4: Algorithm of LZ-77 Encoding

| Algorithm 4:**Deflate** |
|---|
| 1: The whole input stream is divided into series of blocks.<br>2: LZ77 is implemented to find the strings repeated in each block and reference pointers are inserted within the block.<br>3: Symbols are then replaced with the code-words based on their occurrences in the block using Huffman Encoding process. |

Fig 5: Algorithm of Deflate

| Algorithm 5:**LZMA** |
|---|
| 1: Input sequence is passed to a Delta filter which transmits the data in the form of difference between the adjacent symbols. Thus the output bit is the difference between the current bit and the previous bit.<br>2:The output of the Delta filter is encoded by sliding window encoder that uses a search buffer to make references as in the case of LZ-77.<br>3: Range encoder encodes the symbols with values based on the probability distribution of each symbol. The range encoder consist of these steps:<br>e. A large integral range is selected.<br>f. Probability distribution of each input symbol is defined.<br>g. Divide the range among symbols proportional to their probability distribution.<br>h. Each symbol is encoded by dividing the current range till the sub range of the next symbol. |

Fig 6: Algorithm of LZMA

## IV. Experimental Results and Analysis

The algorithms were benchmarked on a machine with Intel(R) i5-3230M processor (clock speed of 2.60 GHz), 12GB of RAM and running Windows 8.1 Professional.

Three well known datasets are used by researchers to compare and study universal lossless data compression algorithms, Calgary corpus [26], Canterbury Corpus [27] and Silesia Corpus [28]. We are using Silesia Corpus to compare compression algorithm as it provides a data set of files that covers the typical file types used now-a-days. The Corpus covers a wide variety of file types including databases and images with file size ranging from 6 MB to 52 MB. Hence, the Corpus covers the majority of files shared over the internet and

| Algorithm 6:**Bzip2** |
|---|
| 1: The input file is traversed from the beginning, reading a single character at a time.<br>2: Run Length Encoding (RLE) is applied on the data.<br>3: Burrow Wheeler technique (BWT) sorts the input by bringing together characters with similar contexts.<br>4: Every symbol is replaced by its index location and the symbol is moved to the front of the array. Therefore the long run of symbols are replaced by 1st location index i.e. 0.<br>5: Obtained data is then compressed by Run Length Encoding to minimize the length of long strings of zero.<br>6: Output is subjected to Huffman Encoding. Multiple Huffman tables are formed in a single block since creating a new table provides better efficiency than using an existing table.<br>7: To reconstruct Huffman table each code bit-length is stored as an encoded difference against the previous code bit-length.<br>8: A bitmap is formed to indicate which symbols are used in the Huffman table and which symbols should be included in the Huffman tree. |

Fig 7: Algorithm of Bzip2

| Algorithm 7:**Prediction_By_Partial_Matching** |
|---|
| 1: A Markov model of $n^{th}$ order is assumed.<br>2: The input file (A) is traversed from the beginning, reading a single symbol at a time.<br>3: For symbol $A_i$ , which is at the position i the preceding n symbols are used to update the probability of the sequence B $(A_{i-n},A_{i-n-1}\ldots A_{i-1},A_i)$.<br>4: Assuming B to be a symbol in the target alphabet, update all the necessary structures.<br>5: If any of the symbol in B is not present then either $A_i$ is sent uncoded or B is shortened and then added to the alphabet. |

Fig 8: Algorithm of Prediction by Partial Matching

therefore is a good data set to compare the performance of compression algorithms. The files present in the Corpus are detailed in Table 1.

Firstly, we analyse each of the algorithms separately and their performance on the Silesia Corpus. The results of performance of the algorithms are recorded in table 2-6. Further, we compare the algorithms on the basis of their compression ratios, compression speed and decompression speed.

### *A. Evaluating the Performance of Algorithms*

*1) Deflate*: For Deflate, popular compression program, GZIP (ver. 1.2.4) [17] is used for the test. Option -9 is used for best possible compression. Experiments were performed on Silesia Corpus and the results were recorded in the table 2.

*2) LZMA:* For LZMA, popular compression program, 7zip (ver. 16.02) [16] is used for the test. Option -a1 is used for maximum compression mode. Moreover, option -d27 is used which sets the dictionary size as 27 MB. Experiments were performed on Silesia Corpus and the results were recorded in the table 3.

*3) Bzip2:* Bzip2 (ver. 1.5) [21] is used for testing, with option -9 for maximum performance. Option -9, sets the blocksize to the maximum limit i.e. 900kB. Experiments were performed on Silesia Corpus and the results were recorded in the table 4.

*4) PPM:* For PPM, compression program PPMd (variant J rev. 1) [29] and PPMontsr (variant J rev.1) [29] are used. PPMduses the PPM compression algorithm along with Information Inheritance [30]. PPMonstr is a slower, but better performing version of PPMd.

*a) PPMd:* Option -o16 is used to set the order of the PPM to 16. Option -m is used to select the memory limit which is set to be 10 times the size of the file to be compressed (limited to 256 MB, by the program). Experiments were performed on Silesia Corpus and the results were recorded in the table 5.

*b) PPMonstr:* Option -o16 is used to set the order of the PPM to 16. Option -m is used to select the memory limit which is set to be 10 times the size of the file to be compressed. Experiments were performed on Silesia Corpus and the results were recorded in the table 6.

TABLE I. DESCRIPTION OF FILES ON SILESIA CORPUS

| File Name | Data Type | Size (Bytes) | Description |
|---|---|---|---|
| dickens | Text in English | 10,192,446 | Collected works of Charles Dickens (from Project Gutenberg) |
| mozilla | Executable file | 51,220,480 | Tarred executables of Mozilla 1.0 (Tru64 Unix edition) (from Mozilla Project) |
| mr | Image file | 9,970,564 | Medical magnetic resonance image |
| nci | Database | 33,553,445 | Chemical database of structures |
| ooffice | Executable file | 6,152,192 | A dynamic linked library from OpenOffice.org 1.01 |
| osdb | Database | 10,085,684 | Sample database in MySQL format from Open Source Database Benchmark |
| reymont | PDF file | 6,625,583 | Text of the book Chłopi by Władysław Reymont |
| samba | Executable file | 21,606,400 | Tarred source code of Samba 2-2.3 (from Samba Project) |
| sao | Binary Database | 7,251,944 | The SAO star catalogue (from Astronomical Catalogues and Catalogue Formats) |
| webster | HTML file | 41,458,703 | The 1913 Webster Unabridged Dictionary (from Project Gutenberg) |
| xml | XML file | 5,345,280 | Collected XML files |
| x-ray | Image file | 8,474,240 | X-ray medical picture |

TABLE II. PERFORMANCE OF DEFLATE ON SILESIA CORPUS

| File Name | Original Size (Bytes) | Final Size (Bytes) | Comp. Ratio | Comp. Speed (MB/s) | Decomp. Speed (MB/s) |
|---|---|---|---|---|---|
| dickens | 10,192,446 | 3,851,823 | 2.646 | 8.572 | 72.002 |
| mozilla | 51,220,480 | 18,994,142 | 2.697 | 4.958 | 115.207 |
| mr | 9,970,564 | 3,673,940 | 2.714 | 4.999 | 98.028 |
| nci | 33,553,445 | 2,987,533 | 11.231 | 10.832 | 195.116 |
| ooffice | 6,152,192 | 3,090,442 | 1.991 | 7.512 | 40.186 |
| osdb | 10,085,684 | 3,716,342 | 2.714 | 14.798 | 102.324 |
| reymont | 6,625,583 | 1,820,834 | 3.639 | 4.264 | 107.096 |
| samba | 21,606,400 | 5,408,272 | 3.995 | 13.885 | 142.107 |
| sao | 7,251,944 | 5,327,041 | 1.361 | 9.296 | 82.333 |
| webster | 41,458,703 | 12,061,624 | 3.437 | 9.229 | 34.866 |
| xml | 5,345,280 | 662,284 | 8.071 | 13.522 | 37.760 |
| x-ray | 8,474,240 | 6,037,713 | 1.404 | 14.911 | 69.074 |
| **Average** | - | - | **3.825** | **9.732** | **91.342** |

TABLE III. PERFORMANCE OF LZMA ON SILESIA CORPUS

| File Name | Original Size (Bytes) | Final Size (Bytes) | Comp. Ratio | Comp. Speed (MB/s) | Decomp. Speed (MB/s) |
|---|---|---|---|---|---|
| dickens | 10,192,446 | 2,830,389 | 3.601 | 1.185 | 60.374 |
| mozilla | 51,220,480 | 13,362,705 | 3.833 | 1.643 | 53.095 |
| mr | 9,970,564 | 2,750,927 | 3.624 | 1.677 | 48.762 |
| nci | 33,553,445 | 1,529,702 | 21.934 | 1.440 | 193.934 |
| ooffice | 6,152,192 | 2,426,876 | 2.535 | 1.983 | 30.088 |
| osdb | 10,085,684 | 2,841,980 | 3.548 | 1.555 | 49.837 |
| reymont | 6,625,583 | 1,314,746 | 5.039 | 1.216 | 57.969 |
| samba | 21,606,400 | 3,740,983 | 5.775 | 1.964 | 82.094 |
| sao | 7,251,944 | 4,423,369 | 1.639 | 2.076 | 23.848 |
| webster | 41,458,703 | 8,369,031 | 4.953 | 1.065 | 31.404 |
| xml | 5,345,280 | 438,203 | 12.198 | 2.231 | 36.940 |
| x-ray | 8,474,240 | 4,491,727 | 1.886 | 1.996 | 24.564 |
| **Average** | - | - | **5.88** | **1.669** | **57.742** |

TABLE IV. PERFORMANCE OF BZIP2 ON SILESIA CORPUS

| File Name | Original Size (Bytes) | Final Size (Bytes) | Comp. Ratio | Comp. Speed (MB/s) | Decomp Speed (MB/s) |
|---|---|---|---|---|---|
| dickens | 10,192,446 | 2,799,520 | 3.641 | 7.612 | 10.319 |
| mozilla | 51,220,480 | 17,914,392 | 2.859 | 8.686 | 21.396 |
| mr | 9,970,564 | 2,441,280 | 4.084 | 12.628 | 22.857 |
| nci | 33,553,445 | 1,812,734 | 18.510 | 4.300 | 24.059 |
| ooffice | 6,152,192 | 2,862,526 | 2.149 | 8.194 | 15.440 |
| osdb | 10,085,684 | 2,802,792 | 3.598 | 8.430 | 15.691 |
| reymont | 6,625,583 | 1,246,230 | 5.316 | 8.691 | 17.311 |
| samba | 21,606,400 | 4,549,759 | 4.749 | 8.971 | 25.789 |
| sao | 7,251,944 | 4,940,524 | 1.468 | 6.433 | 12.091 |
| webster | 41,458,703 | 8,644,714 | 4.796 | 6.858 | 14.671 |
| xml | 5,345,280 | 441,186 | 19.208 | 4.988 | 18.013 |
| x-ray | 8,474,240 | 4,051,112 | 1.320 | 8.940 | 14.380 |
| **Average** | - | - | **5.975** | **7.894** | **17.668** |

TABLE V. PERFORMANCE OF PPMD ON SILESIA CORPUS

| File Name | Original Size (Bytes) | Final Size (Bytes) | Comp. Ratio | Comp. Speed (MB/s) | Decomp. Speed (MB/s) |
|---|---|---|---|---|---|
| dickens | 10,192,446 | 2,327,178 | 4.380 | 3.542 | 3.377 |
| mozilla | 51,220,480 | 15,682,535 | 3.266 | 3.389 | 3.086 |
| mr | 9,970,564 | 2,335,937 | 4.268 | 4.483 | 4.222 |
| nci | 33,553,445 | 1,834,358 | 18.292 | 22.889 | 21.447 |
| ooffice | 6,152,192 | 2,496,315 | 2.464 | 2.945 | 2.695 |
| osdb | 10,085,684 | 2,354,180 | 4.284 | 3.762 | 3.608 |
| reymont | 6,625,583 | 1,052,589 | 6.295 | 5.192 | 4.860 |
| samba | 21,606,400 | 3,720,363 | 5.808 | 6.338 | 5.765 |
| sao | 7,251,944 | 4,759,281 | 1.524 | 1.775 | 1.565 |
| webster | 41,458,703 | 6,666,916 | 6.219 | 4.854 | 4.437 |
| xml | 5,345,280 | 377,581 | 14.157 | 15.930 | 11.746 |
| x-ray | 8,474,240 | 3,874,128 | 2.187 | 2.388 | 2.163 |
| **Average** | - | - | **6.095** | **6.457** | **5.748** |

## B. Comparison of Algorithms

*1) Average Compression Ratio:* Most of the time while sending files over the network, the network bandwidth is limited. Hence, it is necessary to reduce the size of the file as much as possible. Compression ratio provides an idea about the capability of a compression algorithm in compressing wide varieties of file shared over the network [12]. Compression ratio is defined as (1),

$$Compression\ Ratio = \frac{Uncompressed\ File\ Size}{Compressed\ File\ Size} \quad (1)$$

TABLE VI. PERFORMANCE OF PPMONSTR ON SILESIA CORPUS

| File Name | Original Size (Bytes) | Final Size (Bytes) | Comp. Ratio | Comp. Speed (MB/s) | Decomp. Speed (MB/s) |
|---|---|---|---|---|---|
| dickens | 10,192,446 | 2,175,080 | 4.686 | 0.726 | 0.734 |
| mozilla | 51,220,480 | 11,960,919 | 4.282 | 0.468 | 0.463 |
| mr | 9,970,564 | 2,231,540 | 4.468 | 0.752 | 0.750 |
| nci | 33,553,445 | 1,451,252 | 23.120 | 2.202 | 2.195 |
| ooffice | 6,152,192 | 1,996,223 | 3.082 | 0.428 | 0.421 |
| osdb | 10,085,684 | 2,244,487 | 4.494 | 0.603 | 0.602 |
| reymont | 6,625,583 | 906,181 | 7.312 | 0.936 | 0.926 |
| samba | 21,606,400 | 3,142,854 | 6.875 | 0.719 | 0.714 |
| sao | 7,251,944 | 4,047,838 | 1.792 | 0.256 | 0.253 |
| webster | 41,458,703 | 5,819,779 | 7.124 | 0.851 | 0.858 |
| xml | 5,345,280 | 343,552 | 15.559 | 1.652 | 1.631 |
| x-ray | 8,474,240 | 3,756,717 | 2.256 | 0.372 | 0.374 |
| **Average** | - | - | **7.088** | **0.830** | **0.827** |

The average compression ratios of the algorithms are compared in fig. 9. The compression ratios of the algorithms with respect to a file are compared in fig. 10 and fig. 11. From the figures 9-11, we observe that PPM based algorithms are the best in terms of compression ratio. PPMonstr provides the best average compression ratio amongst the algorithms compared with PPMd as the distant second. LZMA and Bzip2 provide moderate level of compression as compared to PPM based algorithms, with Bzip2 providing slighly better results in comparison to LZMA. Deflate provides lowest average compression ratio.

*2) Average Compression Speed:* In many applications, such as when we attach a media file in mail, that file needs to be compressed and stored over the mail server as soon as possible. Hence, in such cases compression speed is extremely important [13]. So, it becomes necesesary to compare the algorithms on the basis of compression speed as well. Compression speed is defined as (2),

$$Compression\ Speed\ (MB/s) = \frac{Uncompressed\ File\ Size\ (MB)}{Compression\ Time\ (secs)} \quad (2)$$

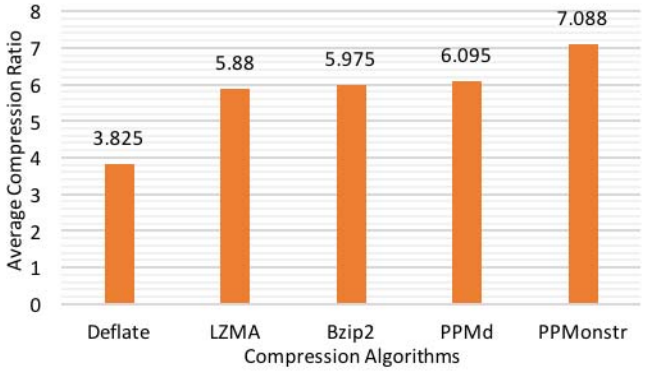


Fig 9. Comparison of Average Compression Ratios

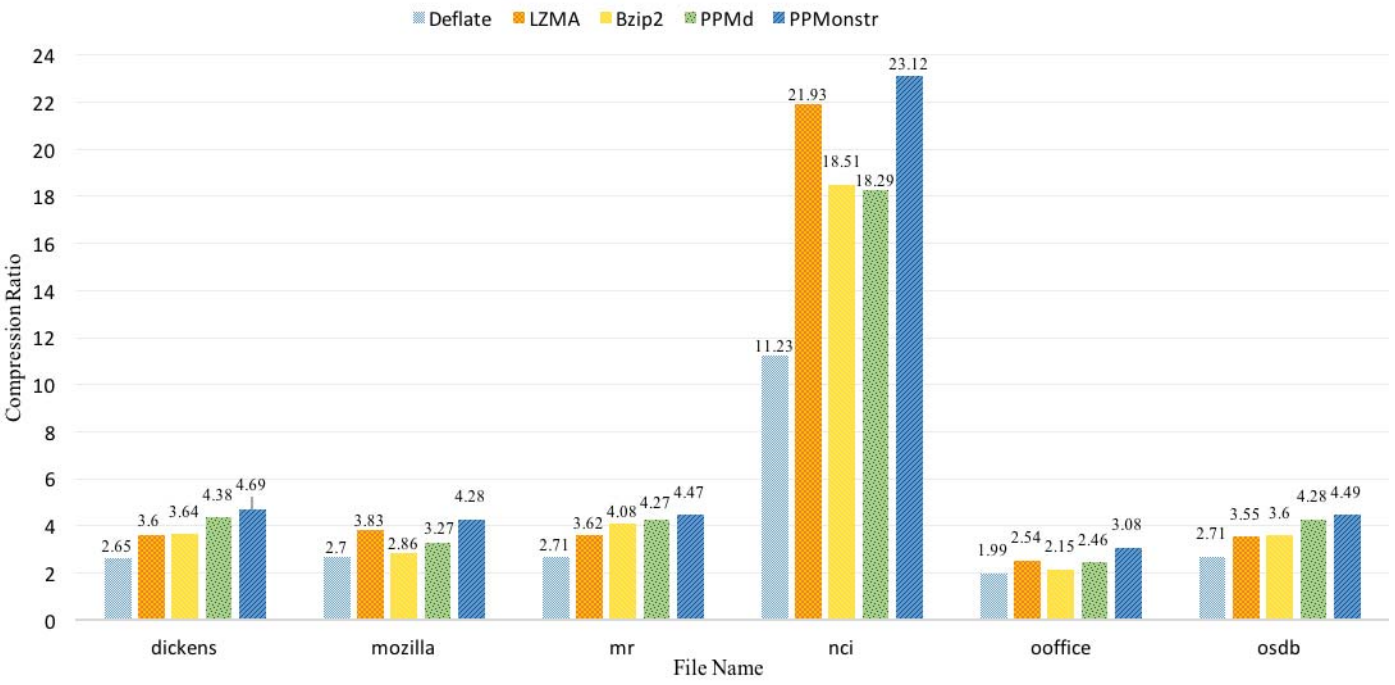


Fig. 10 : Comparison of Compression Ratios

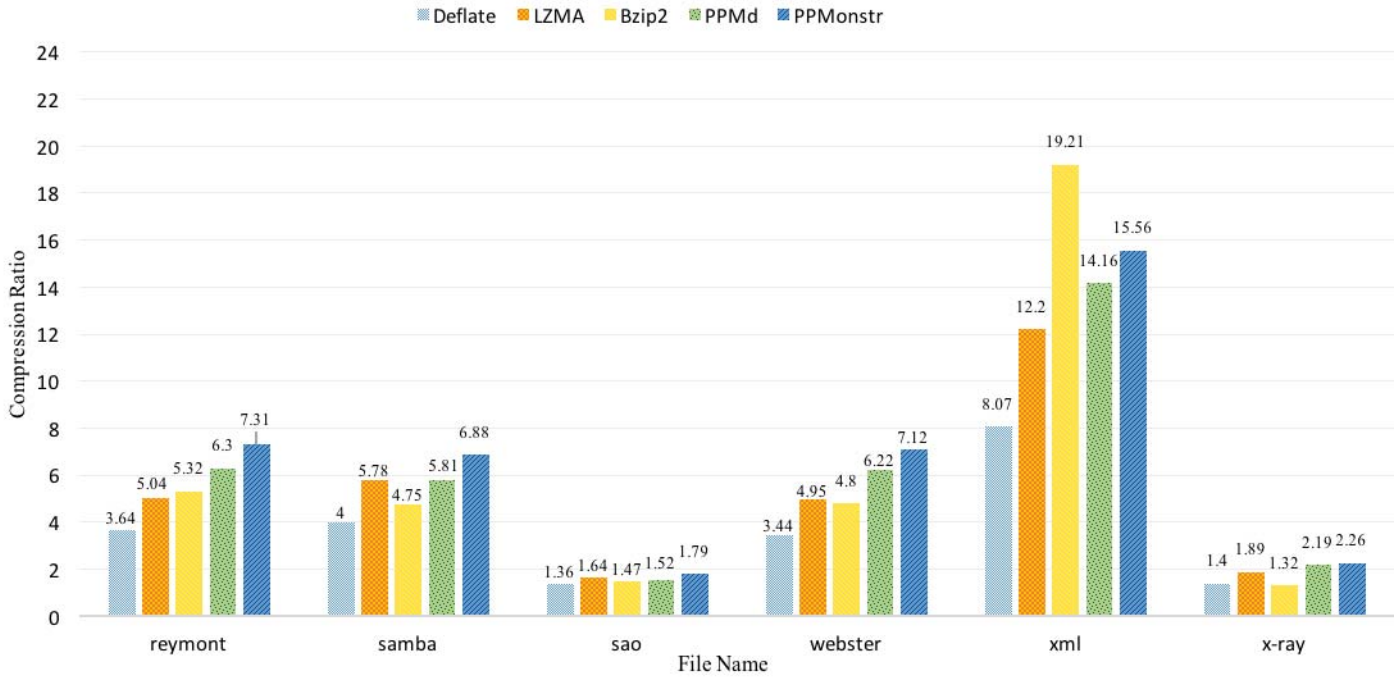


Fig. 11 : Comparison of Compression Ratios

The average compression speed of the algorithms are compared in fig. 12. We observed that Deflate has the highest compression speed but provides a low compression ratio. Bzip2 and PPMd provide moderate compression speed and a good compression ratio as well. Therefore, Bzip2 and PPMd are considered as good choices, when along with compression ratio, compression speed is also an important factor in deciding the compression algorithm. LZMA and PPMonstr have very low compression speed and would find little application in cases where compression speed is of utmost priority.

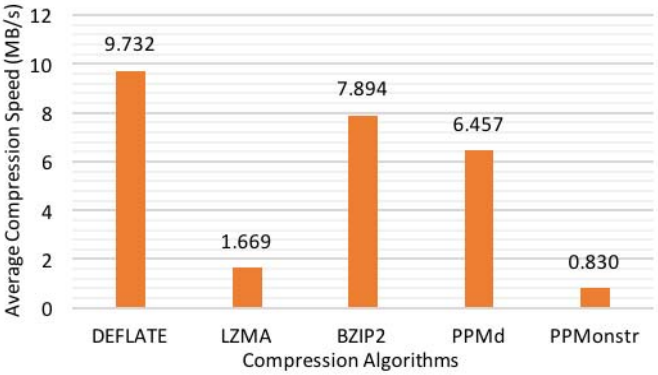


Fig 12. Comparison of Average Compression Speed

*3) Average Decompression Speed:* While buffering videos online, the data coming from server is in compressed state, and hence, it is necessary to decompress the data first. Moreover, for smooth playback of the video, the algorithm should be able to decompress the data in limited amount of time. Hence, decompression speed of the algorithm needs to be considered for such algorithms [14]. Decompression speed is defined as (3),

$$Decompression\ Speed\ (MB/s) = \frac{Uncompressed\ File\ Size\ (MB)}{Uncompression\ Time\ (secs)} \quad (3)$$

The average decompression speed of the algorithms are compared in fig. 13. The results reveals that Deflate provides the best average decompression speed LZMA provides moderate decompression speed but coupled with high compression ratios, it is an ideal algorithm for applications requiring high compression ratio along with fast decompression speeds. Bzip2 provides low to moderate decompression speeds. PPMd and PPMonstr provide very low decompression speeds which make them useless for tasks demanding high decompression speeds.

## V. Conclusion and Future Work

In this work we have compared modern lossless compression algorithm by evaluating their performance on Silesia corpus. After analyzing the compression ratio, compression speed and decompression speed of the algorithms, we conclude the following:

A) Deflate provides poor compression ratio but has the fastest compression and decompression speed.

B) LZMA provides good compression ratio along with high decompression speed and hence is well suited for applications dependent on the both the factors, such as online video playback.

C) Bzip2 provides good compression ratio along with high compression speed and is an ideal algorithm for applications requiring both the factors, such as attaching a file in mail.

D) PPM based algorithms, PPMd and PPMonstr, provide very high compression ratios but have very low compression and decompression speed. They are suitable only when the network bandwidth is very limited and very high compression is needed.

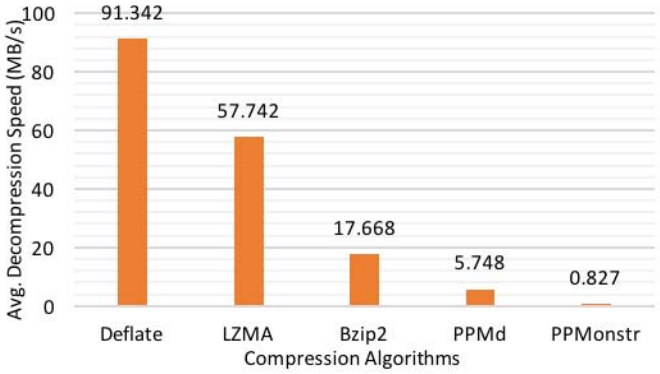


Fig 13. Comparison of Average Decompression Speed

As discussed earlier, Silesia Corpus covers wide variety of files like text (dickens and reymont), image (mr and x-ray), database (nci, osdb and sao), web files (webster and xml) and executable files (mozilla, ooffice and samba). Therefore, the corpus contains most ofthe file types that are shared over the internet. However, from the experimental results, it is observed that there isn't any compression algorithm that performs optimally for all file types. Some compression algorithms are better than the others in terms of compression ratio while some in terms of compression and decompression speed. Hence, it becomes necessary to decide the compression algorithm on the basis of their file type and the requirements of the application.

In future, Calgary corpus and Canterbury corpus can also be used to compare the compression algorithms. New datasets can also be created which better represent the file types shared over the internet. Moreover, we can extend our analysis to include recently developed compression algorithms such as snappy [31] and cmix [32].As we become more dependent on internet, the need for more advanced compression algorithmsis felt and hence, the focus should be on developing more efficient compression algorithms with better compression ratio and speed.

## References

[1] JoshJames,DOMO, Data never sleeps 3.0, https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/Last visited 21 April 2016.

[2] Blelloch, E., Introduction to Data Compression, Computer Science Department, Carnegie Mellon University, 2002. [Online]. Available: https://www.cs.cmu.edu/~guyb/realworld/compression.pdf

[3] Ziv, Jacob; Lempel, Abraham. "A Universal Algorithm for Sequential Data Compression". IEEE Transactions on Information Theory**23** (3): pp. 337–343, May 1977.

[4] Huffman, D."A Method for the Construction of Minimum-Redundancy Codes". Proceedings of the IRE**40** (9): pp. 1098–1101, 1952.

[5] JPEG, Joint Photographic Experts Group, JPEG Homepage, https://jpeg.org/ . Last visited 21 April 2016.

[6] Bismita Gadanayak,Chittaranjan Pradhan, "Selective Encryption of MP3 Compression", in Proc. International Conference on Information Systems and Technology (ICIST) 2011.

[7] Burns, R. W."Communications: an international history of the formative years", Institution of Electrical Engineers,2004.

[8] Shannon, C.E. "A Mathematical Theory of Communication" Bell System Technical Journal**27**,pp. 379–423, July 1948.

[9] Fano, R.M. "The transmission of information". Technical Report No. 65, USA: Research Laboratory of Electronics at MIT, 1949.

[10] Ziv, Jacob; Lempel, Abraham, "Compression of Individual Sequences via Variable-Rate Coding". IEEE Transactions on Information Theory**24** (5): 530–536, September 1978.

[11] Welch, Terry. "A Technique for High-Performance Data Compression". Computer **17** (6): 8–19, 1984.

[12] Mohammed Al-laham, Ibrahiem M. M. El Emary, "Comparative Study Between Various Algorithms of Data Compression Techniques", in Proc. World Congress on Engineering and Computer Science, 2007

[13] S.R. Kodituwakku, U.S. Amarasinghe . "Comparison of Lossless Data Compression Algorithm for Text Data", Indian Journal of Computer Science and Engineering Vol **1** (4) : 416-426, 2010

[14] Arup Kumar Bhattacharjee, Tanumon Bej, Saheb Agarwa. "Comparison Study of Lossless Data Compression Algorithms for Text Data" ,IOSR Journal of Computer Engineering (IOSR-JCE) Volume **11** (6) : 15-19, 2013

[15] Senthil Shanmugasundaram, Robert Lourdusamy "A Comparative Study of Text Compression Algorithm", International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011

[16] I. Pavlov ,2002, 7-Zip, http://www.7-zip.org/. Last visited 12 september 2016

[17] J. L. Gailly, 1993, The gzip program, http://www.gzip.org/. Last visited 12 September 2016

[18] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," IEEE Trans. Commun., vol. COM-32, pp. 396-402, April 1984.

[19] Igor Pavlov, "7z format", http://www.7zip.org/7z.html. Last visited 10 Spetember 2016

[20] Ranganathan, N and Henriques, S. "High-speed VLSI designs for Lempel-Ziv-based data compression". IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol-40, No.2, pp. 96–106, Feb. 1993.

[21] J. Seward, 1996, The bzip2 program, http://www.bzip.org/. Last visited 10 September 2016

[22] P. Deutsch (1996) "DEFLATE Compressed Data Format Specification version 1.3" [Online]. Available: https://tools.ietf.org/html/rfc1951.

[23] Katz, Phillip W. ,"String searcher, and compressor using same", US patent 5051745, Sept. 09, 1991.

[24] E.Jebamalar Leavline ,D.Asir Antony Gnana Singh (2013, Mar) "Hardware Implementation of LZMA Data Compression Algorithm",International Journal of Applied Information Systems (IJAIS), [Online] Volume5-No.4.Available: http://research.ijais.org/volume5/number4/ijais12-450900.pdf

[25] Michael Burrows, David Wheeler, "A Block-sorting Lossless Data Compression Algorithm", Digital Systems Research Center, Research Report 124, 1994.

[26] The Calgary corpus, http://www.data-compression.info/Corpora/ , Last visited 12 September 2016

[27] The Canterbury corpus, http://corpus.canterbury.ac.nz/, Last visited 12 September 2016

[28] The Silesia corpus, http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia, Last visited 21 April 2016

[29] W. J. Teahan, 1997, The ppmd+ program ftp://ftp.cs.waikato.ac.nz/pub/compression/ppm/ppm.tar.gz. Last visited 21 April 2016.

[30] Shkarin, Dmitry, "PPM: One step to practicality", Proc. 2002 Data Compression ConferenceI. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, pp. 271–350,1963.

[31] Snappy, a fast compressor/ decompressor program, https://github.com/google/snappy/blob/master/README. Last visited 13 September 2016

[32] Byron Knoll, cmix program, http://www.byronknoll.com/cmix.html. Last visited 13 September 2016