ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 6, June 2016

A SURVEY ON LOSSLESS TEXT DATA COMPRESSION TECHNIQUES

Mamta Rani and Vikram Singh

***Abstract*- Data Compression refers to the process of reducing the data size and removing the excessive information. The main objective of data compression is to reduce the amount of redundant information in the stored or communicated data. Data compression is quite useful as it helps us to reduce the resources usage such as data storage space or transmission capacity. It finds its application in the area of file storage and distributed system because in distributed system we need to send data from and to all the systems. Data compression techniques are mainly used for speed and performance efficiency along with maintaining the cost of transmission. There are number of different data compression methodologies, which are used to compress different data formats like text, video, audio, image files. Data compression techniques can be broadly classified into two major categories, "lossy" and "lossless" data compression techniques as in [1]. In this paper, reviews of different basic lossless text data compression methods are considered and a conclusion is drawn on the basis of these methods.**

***Index Terms*- Data Compression, Huffman Coding, Lossless data compression, Lossy data compression, Shannon - Fano coding.**

I. INTRODUCTION

Compression is the art of representing the information in a compact form rather than its original or uncompressed form as in [2]. In other words, using the data compression, the size of a particular file can be reduced. This is very useful when processing, storing or transferring a huge file, which needs lots of resources. Data compression is a process by which a file (Text, Audio, Video) may be transformed to another (compressed) file, such that the original file may be fully recovered from the original file without any loss of actual information. When data compression is used in a data transmission application, speed is the primary goal. Speed of transmission depends upon the number of bits sent, the time required for the encoder to generate the coded message and the time required for the decoder to recover the original ensemble. In a data storage application, the degree of compression is the primary concern. So, we have discussed here the different data compression techniques.

II. DATA COMPRESSION TECHNIQUES

Data compression methods can be classified in several ways. One of the most important criteria of classification is whether the compression algorithms remove some part of data which can or cannot be recovered during decompression.

A. Lossy Data Compression

The algorithm which removes some part of data is called lossy data compression. The lossy data compression algorithm is usually used when a perfect consistency with the original data is not necessary after decompression. The examples of frequent use of Lossy data compression are on the Internet and especially in the streaming media and telephony applications. Some examples of lossy data compression algorithms are JPEG, MPEG, and MP3. 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.

B. Lossless data compression

Lossless data compression is a technique that allows the use of data compression algorithms to compress the text 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. Various lossless data compression algorithm have been proposed and used. Some of main techniques are Huffman Coding, Run Length Encoding, Arithmetic Encoding and Dictionary Based Encoding.

III. EXISTING LOSSLESS DATA COMPRESSION TECHNIQUES

The existing data compression techniques are described as follows:

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 method reduces the size of a string considerably when the string is lengthy and the compression ratio is not affected by the content of the string . Bit Reduction Algorithm in steps-

1. Select the frequently occurring characters from the text file which are to be encoded and obtain their corresponding ASCII code.

2. Obtain the corresponding binary code of these ASCII key codes for each character.
3. Then put these binary numbers into an array of byte (8bit array).
4. Remove extra bits from binary number like extra 3 bits from the front.
5. Then rearrange these into array of byte and maintain the array.
6. Final text will be encoded and as well as compression will be achieved.
7. 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 as in [3]. A Huffman code can be determined by successively constructing a binary tree, whereby the leaves represent the characters that are to be encoded. Every node contains the relative probability of occurrence of the characters belonging to the sub tree beneath the node. The edges are labeled with the bits 0 and 1. 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 as in [3].

C. Run length Encoding

Run Length Encoding or simply RLE is the simplest of the data compression algorithms. The consecutive sequences of symbols are identified as runs and the others are identified as non runs in this algorithm. This algorithm deals with some sort of redundancy as in [4]. It checks whether there are any repeating symbols or not, and is based on those redundancies and their lengths. Consecutive recurrent symbols are identified as runs and all the other sequences are considered as non-runs. For an example, the text “ABABBBBC” is considered as a source to compress, then the first 3 letters are considered as a non-run with length 3, and the next 4 letters are considered as a run with length 4 since there is a repetition of symbol B. The major task of this algorithm is to identify the runs of the source file, and to record the symbol and the length of each run. The Run Length Encoding algorithm uses those runs to compress the original source file while keeping all the non-runs without using for the compression process.
RLE is useful where redundancy of data is high or it can also be used in combination with other compression techniques also.
Here is an example of RLE:

Input: YYYBBCCCCDEEEEEERRRRRRRRRR

Output: 3Y2B4C1D6E10R

The drawback of RLE algorithm is that it cannot achieve the high compression ratios as compared to another advanced compression methods, but the advantage of RLE is that it is easy to implement and quick to execute thus making it a good alternative for a complex compression algorithm.

D. Shannon-Fano coding

This is one of an earliest technique for data compression that was invented by Claude Shannon and Robert Fano as in [5] in 1949.In this technique, a binary tree is generated that represent the probabilities of each symbol occurring. The symbols are ordered in a way such that the most frequent symbols appear at the top of the tree and the least likely symbols appear at the bottom.

The algorithm for Shanon- Fano coding 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.
5. Divide the list in two while keeping the probability of the left branch roughly equal to those on the right branch.
6. Prepend 0 to the left node and 1 to the right node codes.
7. Recursively apply steps 5 and 6 to the left and right sub trees until each node is a leaf in the tree.

E. Arithmetic Coding

Arithmetic coding is an optimal entropy coding technique as it provides best compression ratio and usually achieves better results than Huffman Coding. It is quite complicated as compared to the other coding techniques. When a string is converted in to arithmetic encoding, the characters having maximum probability of occurrence will be stored with fewer bits and the characters that do not occur so frequently will be stored with more bits, resulting in fewer bits used overall. Arithmetic coding converts the stream of input symbols into a single floating point number as output as in [5]. Unlike Huffman coding, arithmetic coding does not code each symbol separately. Each symbol is instead coded by considering all prior data. Thus a data stream encoded in this fashion must always be read from the beginning. Here is an algorithm to generate the arithmetic code:

1. Calculate the number of unique symbols in the input. This number represents the base b (e.g. base 2 is binary) of the arithmetic code.
2. Assign values from 0 to b to each unique symbol in the order they appear.
3. Using the values from step 2, the symbols are replaced with their codes in the input.
4. Convert the result from step 3 from base b to a sufficiently long fixed-point binary number to preserve precision.
5. Record the length of the input string somewhere in the result as it is needed for decoding.

F. *Lempel-Ziv-Welch (LZW) Algorithm*

Dictionary based compression algorithms are based on a dictionary instead of a statistical model. A dictionary is a set of possible words of a language, and is stored in a table like structure and used the indexes of entries to represent larger and repeating dictionary words. The Lempel-Zev Welch algorithm or simply LZW algorithm is one of such algorithms. In this method, a dictionary is used to store and index the previously seen string patterns. In the compression process, those index values are used instead of repeating string patterns. The dictionary is created dynamically in the compression process and no need to transfer it with the encoded message for decompressing. In the decompression process, the same dictionary is created dynamically. Therefore, this algorithm is an adaptive compression algorithm.

G. *Burrows-Wheeler Transform*

Burrows-Wheeler transform does not compress a message, but rather transform it into a form that is more amenable to compression i.e. it can be more efficiently coded by a Run-Length Encoder or other secondary compression technique. The transform rearranges the characters in the input so that that there are lots of clusters with repeated characters, but in a way so that it is still possible to recover the original input. Burrows-wheeler transform (BWT) works in block mode while others mostly work in streaming mode as in [6].
This algorithm classified into transformation algorithm because the main idea is to rearrange (by adding and sorting) and concentrate symbols. These concentrated symbols then can be used as input for another algorithm to achieve good compression ratios.

Since the BWT operates on data in memory, you may encounter files too big to process in one fell swoop. In these cases, the file must be split up and processed a block at a time as in [7]. To speed up the sorting process, it is possible to do parallel sorting or using larger block of input if more memory is available.

The algorithm for a BWT is:

1. The first step is to create a string array.
2. Then generate all the possible rotations of the input string and store each in the array.
3. Sort the array in alphabetical order.
4. Return the last column of the array.

IV. PERFORMANCE MEASURES FOR DATA COMPRESSION ALGORITHMS

Performance measure is use to find which technique is good according to some criteria. The performance of the compression algorithm can be measured on the basis of different criteria depending upon the nature of the application. The most important thing we should keep in mind while measuring performance is space efficiency. Time efficiency is also an important factor. Since the compression behavior depends on the redundancy of symbols in the source file, it is difficult to measure performance of compression algorithm in general. The performance of data compression depends on the type of data and structure of input source. The compression behavior depends on the category of the compression algorithm: lossy or lossless. Performance evaluation of the algorithm is done using two parameters-Compression Ratio and Saving Percentage.

A. *Compression ratio*

Compression ratio is defined as the ratio of size of the compressed file to the size of the source file.

$$CompressionRatio = (C2/C1)$$

where, C1= Size before compression

C2= Size after compression

B. *Saving Percentage*

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

$$SavingPercentage = ((C1 - C2)/C1) * 100\%$$

where, C1= Size before compression

C2= Size after compression

V. CONCLUSION

In this paper, we have presented a review of LZW, Huffman coding and Shannon-Fano coding, Run Length Encoding, Burrows Wheeler Transform techniques of data compression on English words in terms of compression ratio and saving percentage and all the algorithms show different level of compression. After testing these algorithms, Huffman coding and Shannon Fano coding methodologies are very powerful over LZW. Huffman coding and Shannon Fano coding gives better results and reduces the size of the text.

REFERENCES

[1] Apoorv Vikram Singh, Garima Singh,” A Survey on Different text Data Compression Techniques”, International Journal of Science and Research, Vol. 3, July 2014.

[2] S.R. Kodituwakku, U. S.Amarasinghe,” Comparison of lossless data compression algorithms For text data”, Indian Journal Of Computer Science and Engineering, Vol 1 no 4 416-425.

[3] S. Porwal, Y. Chaudhary, J. Joshi and M. Jain , “ Data Compression Methodologies for Lossless Data and Comparison between Algorithms” International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 2, March 2013.

[4] Pu, I.M., 2006, *Fundamental Data Compression*, Elsevier, Britain.

[5] S. Shanmugasundaram and R. Lourdusamy, “A Comparative Study of Text Compression Algorithms” International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011.

[6] I. M.A.D. Suarjaya, "A New Algorithm for Data Compression Optimization", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No. 8, 2012, pp.14-17.
[7] Nelson, M. 1996. Data compression with Burrows-Wheeler Transform, Dr. Dobb's Journal.

Mamta Rani received the B.Tech degree in Computer Science Engineering from Mahavir Educational and Charitable Trust Group Of Institutes, Punjab Technical University, Punjab in 2013, and pursuing M.Tech from Chaudhary Devi Lal University, Sirsa. She has presented a paper in IDEA Conference. Currently, she is doing her thesis work on a new data compression technique. Her topic of interest is data compression.

Vikram Singh works as a Professor of Computer Science at Chaudhary Devi Lal University Sirsa. He holds Masters in Physics, MCA and Doctorate in Computer Science- all from Kurukshetra University. He has published more than 100 research articles and three books. He has delivered numerous expert and resource person talks at conferences and refresher courses. His research interests include operating systems, system simulation and data mining.