International Journal of Engineering Research and Science & Technology

IJERST

ISSN : 2319-5991
Vol. 4, No. 4
November 2015

www.ijerst.com

Email: editorijerst@gmail.com or editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 4, No. 4, November 2015
© 2015 IJERST. All Rights Reserved

Review Article

LOSSLESS TEXT DATA COMPRESSION USING MODIFIED HUFFMAN CODING – A REVIEW

Harsimran Kaur[1]* and Balkrishan Jindal[1]

Corresponding Author:* ***Harsimran Kaur* ✉ *kharsimran02@gmail.com*

Data Compression is a method to increase the storage capacity by eliminating redundancies that occur in most text files. It converts a string of characters into a new string which have the same data in small length. There are two types of data compression, lossless data compression and lossy data compression. Lossless data compression includes texts and lossy data compression includes image, audio and video etc. There are various techniques used for data compression like Bit Reduction, Arithmetic Coding, Run Length Encoding and Huffman Coding etc.

Keywords: Text data compression, Lossless data compression

INTRODUCTION

Data Compression is the process of encoding data to fewer bits than the original representation, so that it takes less storage space and less transmission time while communicating over a network (Brar and Singh, 2013). Data Compression is possible because most of the real world data is very redundant. It is basically defined as a technique that reduces the size of data by applying different methods that can either be lossy or lossless (Kaur and Goyal, 2013). A compression program is used to convert data from an easy-to-use format to one optimized for compactness. Likewise, an uncompressing program returns the information to its original form.

TYPES OF DATA COMPRESSION

Currently, two basic classes of data compression are applied in different areas. One of these is lossy data compression, which is widely used to compress image data files for communication or archives purposes. The other is lossless data compression that is commonly used to transmit or archive text or binary files required to keep their information intact at any time.

LOSSY DATA COMPRESSION

A lossy data compression method is one where the data retrieves after decompression may not be exactly same as the original data, but is "close enough" to be useful for specific purpose. After one applies lossy data compression to a message, the message can never be recovered exactly as it was before it was compressed. When the compressed message is decoded it does not give back the original message. Data has been lost. Because lossy compression

[1] Yadavindra College of Engineering, Talwandi Sabo, Bathinda, Punjab, India.

cannot be decoded to yield the exact original message, it is not a good method of compression for critical data, such as textual data. In a sound file, for example, the very high and low frequencies, which the human ear cannot hear, may be truncated from the file. Figure 1 shows the compression and decompression process over the network. Figure 2 shows the compression methods, lossless data compression and lossy data compression. The lossless data compression that is commonly used to transmit or archive text or binary files required to keep their nformation intact at any time. Lossy data compression, which is widely used to compress image data files for communication or archives purposes.

Lossless data compression is further divided into Run-Length, Huffman coding and Arithmetic coding. Lossy data compression is divided into Joint Photographic Experts Group (JPEG), Moving Picture Experts Group (MPEG) and MPEG Audio Layer 3 (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. Lossy image compression can be used in

Figure 1: Diagrammatic Representation of Compression [3]

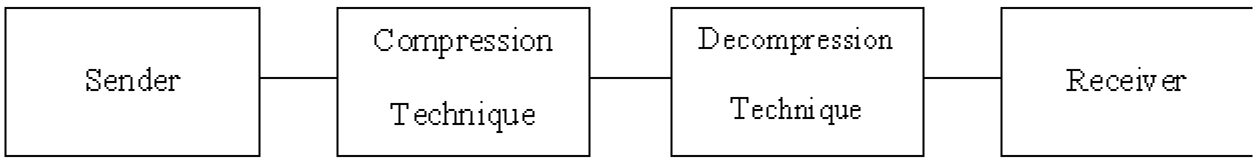
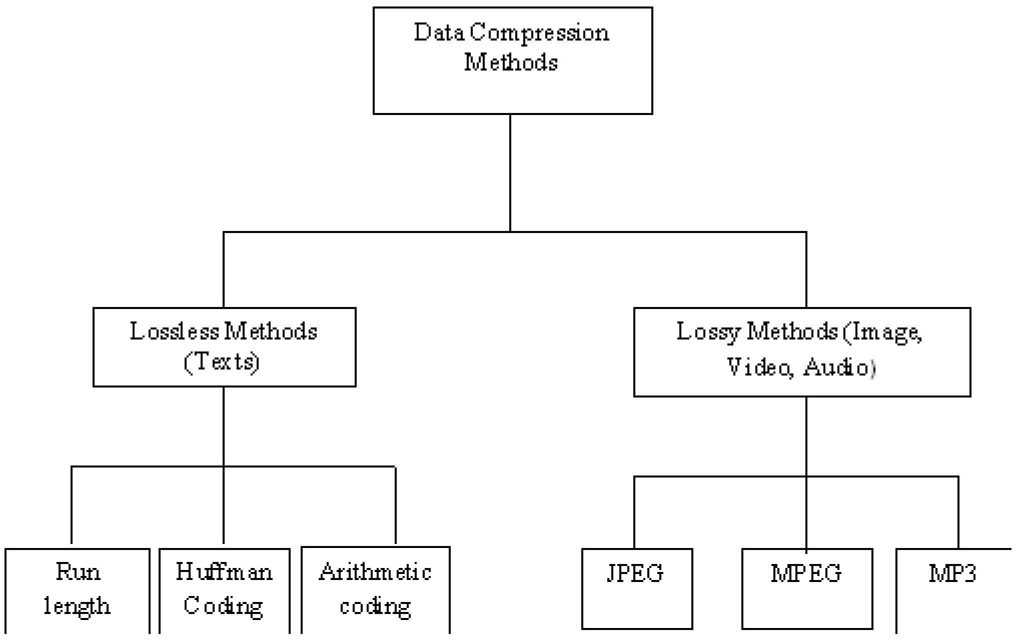


Figure 2: Data Compression Methods [3]



digital cameras to increase storage capacities with minimal degradation of picture quality.

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. It is used when it is important that the original data and the decompressed data be identical. 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.

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 algorithms can be compared using the parameters such as Compression Ratio and Saving Percentage. In a lossless data compression file the original message can be exactly decoded. Lossless data compression works by finding repeated patterns in a message and encoding those patterns in an efficient manner. For this reason, lossless data compression is also referred to as redundancy reduction. Because redundancy reduction is dependent on patterns in the message, it does not work well on random messages. Lossless data compression is ideal for text.

LITERATURE REVIEW

Brar and Singh (2013) described that they provide a survey of different basic lossless and lossy data compression techniques. On the basis of these techniques, a bit reduction algorithm for compression of text data had been proposed by the authors based on number theory system and file differential technique which was a simple compression and decompression technique free from time complexity. The compression algorithm took O(n) time, where n was the total number of characters in the file and the total computation time required for the algorithm was proportional to O(n log n).

Kaur and Goyal (2013) described that they reduced the number of bits required to represent a character by using 6-bit binary coding instead of a 8-bit binary coding technique. They used a numbering map for converting the input data for introduce a way to use the binary form in a dynamic way, which had never used for coding data before, and they further used 6-Digits binary representation of alphabet with lowercase and uppercase with some extra symbols that were most commonly used in the text files. They found a new way to decrease the length of the bits string.

Kaur and Goyal (2013) described that the compression was useful because it help us to reduce the resources usage, such as data storage space or transmission capacity. They discussed only the lossless compression techniques, such as Huffman coding, Run Length Encoding and Arithmetic coding are considered. Lempel Ziv scheme was also considered which was a dictionary based technique. A conclusion was derived on the basis of these methods.

Katugampola (2012) described that how ternary representation of numbers can be utilized to compress text data with fixed-symbol length coding techniques. They used a binary map for ternary digits and introduced a method to use the binary 11-pair, which had never been used for coding data before, and they further used 4-Digits ternary representation of alphabet with lowercase and uppercase letters. They found a way to minimize the length of the bits string, which was only possible in ternary representation.

Islam and Rajon (2011) introduced that the text compression was an elementary concern for data engineering and management. The rapid use of battery powered small memory smart devices especially mobile phones and wireless sensors for communication and monitoring have turned short text compression into a more important and prevailing research arena than large scale text compression. The obtained compression ratio indicated a better performance in terms of resource consumption including better compression ratio, lower compression and decompression time with reduced memory requirements and lower complexity.

Kaur and Verma (2012) described that the LZW was dictionary based algorithm, which was lossless in nature. The code for each character was available in the dictionary which utilized less number of bits (5 bits) than its ASCII code. LZW data compression algorithm was implemented by finite state machine, thus the text data could be effectively compressed.

Shanmugasundaram and Lourdusamy (2011) introduced that there were a lot of data compression algorithms which were available to compress files of different formats. They provide a survey of different basic lossless data compression algorithms. Experimental results and comparisons of the lossless compression algorithms using Statistical compression techniques and Dictionary based compression techniques were performed on text data. Lossy algorithms achieved better compression effectiveness than lossless algorithms, but lossy compression was limited to audio, images, and video, where some loss was acceptable.

Hasan (2011) described that the data compression was of interest in business data processing, both because of the cost savings it offered and because of the large volume of data manipulated in many business applications. More the size of the data be smaller, it provided better transmission speed and saved time.

Mohd Kamir *et al*. (2009) described that the efficiency and capability LZW++ in data compression. The LZW++ technique was enhancement from existing LZW technique. LZW read one by one character at one time. Differ with LZW++ technique, where the LZW++ read three characters at one time. Several experiments had been done by different types of data format. The results showed LZW++ technique was better as compared to existing LZW technique in term of file size. The results showed LZW++ technique was efficient for data compression.

EXISTING LOSSLESS DATA COMPRESSION TECHNIQUES

The existing data compression techniques are described as follow:

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 when the string is lengthy and the compression ratio is not affected by the content of the string (Brara and Singh, 2013). 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 (8 bit array).
4. Remove extra bits from binary no 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.

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 (Kaur and Goyal, 2013). 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. Start with a list of free nodes, where each node corresponds to a symbol in the alphabet.
2. Select two free nodes with the lowest weight from the list.
3. Create a parent node for these two nodes selected and the weight is equal to the weight of the sum of two child nodes.
4. Remove the two child nodes from the list and the parent node is added to the list of free nodes.
5. Repeat the process starting from step-2 until only a single tree remains.

After building the Huffman tree, the algorithm creates a prefix code for each symbol from these alphabets simply by traversing the binary tree from the root to the node, which corresponds to the symbol. It assigns 0 for a left branch and 1 for a right branch.

Run Length Encoding

Data often contains sequences of identical bytes. By replacing these repeated byte sequences with the number of occurrences, a substantial reduction of data can be achieved. This is known as Run-length Encoding. Run-Length Encoding is a simple data compression algorithm which is supported by bitmap file formats such as BMP. RLE basically compresses the data by reducing the physical size of a repeating string of characters. This repeating string is called a run (Kaur and Goyal, 2013) which is typically encoded into two bytes where the first byte represents the total number of characters in the run and is called the run count and it replaces runs of two or more of the same character with a number which represents the length of the run which will be followed by the original character and single characters are coded as runs of 1. Run-length coding is a generalization of zero suppression, which assumes that just one symbol appears particularly often in sequences. The blank (space) character in text is such a symbol; single blanks or pairs of blanks are ignored. Starting with sequences of three bytes, they are replaced by an M-byte and a byte specifying the number of blanks in the sequence. 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.

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 (Kaur and Goyal, 2013). 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. Consequently, random access is not possible. 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 previous step, the symbols are replaced with their codes in the input.
4. Convert the result from previous step 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.

PROPOSED METHODOLOGY

Arithmetic Coding Algorithm

Arithmetic coding transforms the input data into a single rational number between 0 and 1 by changing the base and assigning a single value to each unique symbol from 0 up to the base. Then, it is further transformed into a fixed-point binary number which is the encoded result. The value can be decoded into the original output by changing the base from binary back to the original base and replacing the values with the symbols they correspond to. A general algorithm to compute the arithmetic code is:

1. Calculate the number of unique symbols or characters 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, replace the symbols in the input with their codes.
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.

Example

Step 1: Read the input string.
I am studying in Yadavindra College
Total Characters = 35, Base = 8 bits, Length = 35*8=280 bits

Step 2: Calculate the number of unique symbols or characters in the input.
Unique characters = I amstudyingvrcole = 18
Base = 5 bits

Step 3: Assign values from 0 to 4 to each unique symbol in the order they appear.

I=00000, space=00001, a=00010, m=00011, s=00100, t=00101, u=00110,

d=00111, y=01000, i=01001, n=01010, g=01011, v=01100, r=01101, c=01110,

o=01111, l=10000, e=10001

Step 4: Convert the result from previous step.

00000 00001 00010 00011 00001 00100 00101 00110 00111 01000 01001 01010 01011 00001 01001 01010 00001 01000 00010 00111 00010 01100 01001 01010 00111 01101 00010 00001 01110 01111 10000 10000 10001 01011 10001

Step 5: Record the length of the input string.

Length=35*5=s175 bits

So in this way, we save 105 bits.

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 (Kaur and Goyal, 2013). 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) Start with a list of free nodes, where each node corresponds to a symbol in the alphabet.

(2) Select two free nodes with the lowest weight from the list.

(3) Create a parent node for these two nodes selected and the weight is equal to the weight of the sum of two child nodes.

(4) Remove the two child nodes from the list and the parent node is added to the list of free nodes.

(5) Repeat the process starting from step-2 until only a single tree remains.

Frequencies Characters in the File

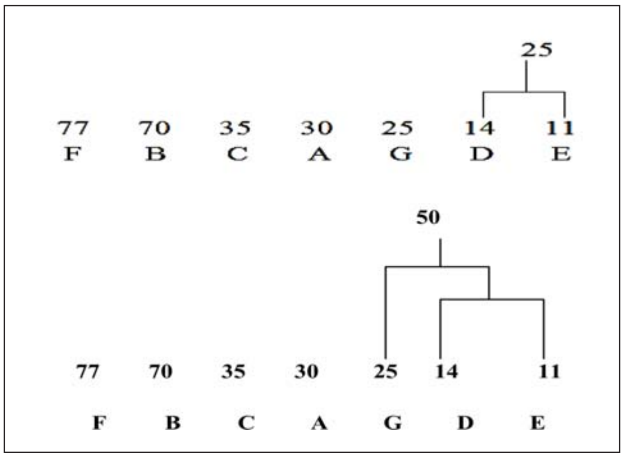
A:	30
B:	70
C:	35
D:	14
E:	11
F:	77
G:	25

After building the Huffman tree, the algorithm creates a prefix code for each symbol from these alphabets simply by traversing the binary tree from the root to the node, which corresponds to the symbol. It assigns 0 for a left branch and 1 for a right branch.

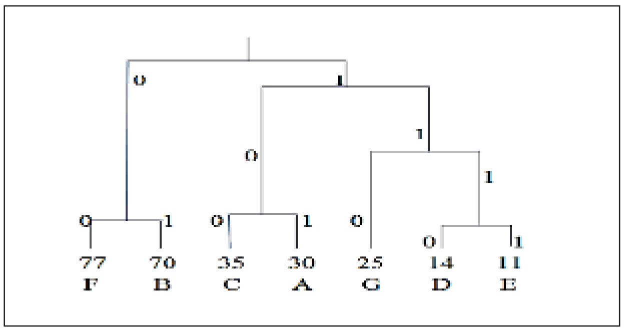
77	70	35	30	25	14	11
F	B	C	A	G	D	E

Example

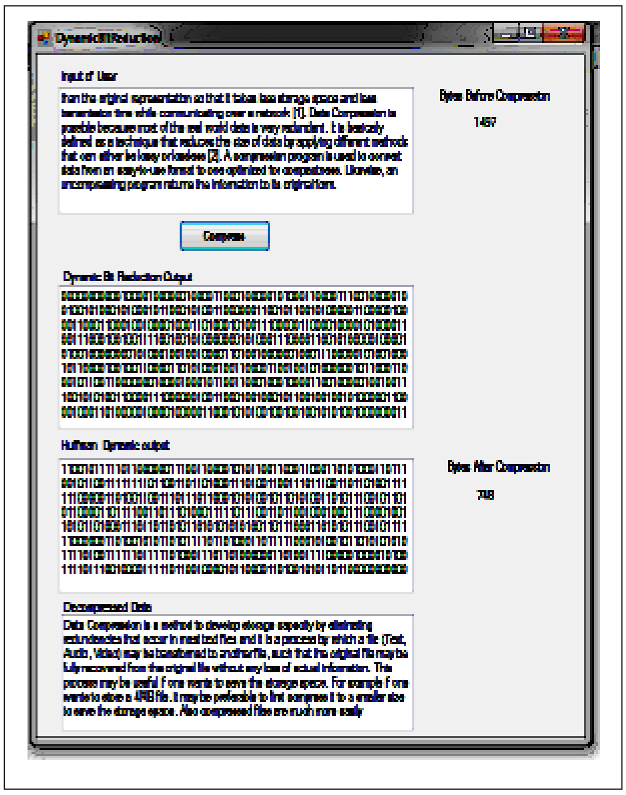
Step 1: In first step of building a Huffman Code order the characters from highest to lowest frequencies of occurrence as follows:



Step 2: Take two least-frequent characters and logically grouped them together, and then their frequencies are added. The D and E characters have grouped together and we have combined frequency are 25.



Step 3: Now labeling the edges from each parent to its left child with the digit 0 and the edge to right child with 1. Now final binary tree will be as follows:



from the original representation so that it takes less storage space and less transmission time while communicating over a network [1]. Data Compression is possible because most of the real world data is very redundant. It is basically defined as a technique that reduces the size of data by applying different methods that can either be lossy or lossless [2]. A compression program is used to convert data from an easy-to-use format to one optimized for compactness. Likewise, an uncompressing program returns the information to its original form.

```
[illegible]
```

```
[illegible]
```

Data Compression is a method to develop storage capacity by eliminating redundancies that occur in most text files and it is a process by which a file (Text, Audio, Video) may be transformed to another file, such that the original file may be fully recovered from the original file without any loss of actual information. This process may be useful if one wants to save the storage space. For example if one wants to store a 4MB file, it may be preferable to first compress it to a smaller size to save the storage space. Also compressed files are much more easily

RESULT

We have tested our algorithm of different texts or strings and accuracy of the algorithm is approximately 93%.

CONCLUSION AND FUTURE WORK

In this paper we have presented the review on various data compression techniques along with their algorithms. It is concluded that a new approach is needed to be developed that can increase the data compression ratio in minimum amount of time.

REFERENCES

1. Brar Rupinder Singh and Singh Bikramjeet. (2013), "A Survey on Different Compression

Techniques and Bit Reduction Algorithm for Compression of Text/Lossless Data", *In: International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 3, No. 3, pp. 579-582.

2. Franceschini Robert and Mukherjee Amar (1996), "Data Compression Using Encrypted Text", *In: IEEE*, pp. 130-138.
3. Hasan Md. Rubaiyat (2011), "Data Compression using Huffman based LZW Encoding Technique", *In: International Journal of Scientific & Engineering Research*, Vol. 2, No. 11, pp. 1-7.
4. https://www.wikipedia.com
5. Islam Md. Rafiqul and Rajon S A Ahsan (2011), "An Enhanced Scheme for Lossless Compression of Short Text for Resource Constrained Devices", In: Proceedings of 14th International Conference on Computer and Information Technology (ICCIT 2011) 22-24 December, 2011, Dhaka, Bangladesh, IEEE.
6. Kaur Rajinder and Goyal Er. Monica (2013), "An Algorithm for Lossless Text Data Compression", *In: International Journal of Engineering Research & Technology*, Vol. 2, No. 7, pp. 474-477.
7. Kaur Rajinder and Goyal Monica (2013), "A Survey on the Different Text Data Compression Techniques", *In: International Journal of Advanced Research in Computer Engineering & Technology,* Vol. 2, No. 2, pp. 711-714.
8. Katugampola Udita N (2012), "A New Technique for Text Data Compression", *In: International Symposium on Computer, Consumer and Control, IEEE*, pp. 405-409.
9. Kaur Simrandeep and Verma V Sulochana (2012), "Design and Implementation of LZW Data Compression Algorithm", *In: International Journal of Information Sciences and Techniques (IJIST)*, Vol. 2, No. 4, pp.71-81.
10. Mohd Kamir Yusof, Mohd Sufian Mat and Ahmad Faisal Amri Abidin (2009), "Study of Efficiency and Capability LZW++ Technique in Data Compression", *In: World Academy of Science, Engineering and Technology,* Vol. 59, pp. 380-383.
11. Porwal Shrusti, Chaudhary Yashi, Joshi Jitendra and Jain Manish (2013), "Data Compression Methodologies for Lossless Data and Comparison between Algorithms", *In: International Journal of Engineering Science and Innovative Technology (IJESIT),* Vol. 2, No. 2, pp. 142-147.
12. Shanmugasundaram Senthil and Lourdusamy Robert (2011), "A Comparative Study of Text Compression Algorithms", *In: International Journal of Wisdom Based Computing*, Vol. 1, No. 3, pp. 68-76.
13. Singh Udepal and Garg Upasna (2013), "An ASCII Value Based Text Data Encryption System", In: *International Journal of Scientific and Research Publications,* Vol. 3, No. 11, pp. 1-5.

ISSN 2319-5991