© 2015 IJSRSET | Volume 1 | Issue 3 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Optimized Compression and Decompression Software

Mohd Shafaat Hussain, Manoj Yadav

Scholar, Department of Computer Science and Engineering, Al-Falah School of Engineering and Technology, Dhauj, Faridabad, Haryana, India

ABSTRACT

This research paper offers data compression practices and matches their performance with the help of a software .In this research paper we are going to discuss the software which compare three algorithms, This is the software which we are going to compare three compression algorithms Huffman, LZW, LZM .With the help these algorithms we capable to compress text file, audio file, video file and images files. The software is basically comparing the algorithms on the basis of space complexity, time complexity of the data after compression. Technically the purpose of the data compression is to reduce duplicity in the stored and communicated data, thus increasing effective data density, Data compression is an important application in file storage and the distributed system. So if we need to increase the performance or heighten the speed so data compression is the advantage at that moment and compressed data is always suitable to reduce the cost both for storage and communication, data compression fundamentally diminishes the size of the data in terms of bits or bytes and the reduced data need lesser disk space compare to the data without compression. We have abundant ways to compress the data like text, audio, video files .Fundamentally we have two distinct characteristics for data compression and they are lossless and lossy compression. In lossless the integrity of the data is always maintained on the other hand In lossy compression data reduce by permanently eliminating certain amount of information, especially duplicate information, when a file in uncompressed, only a part of the original information is still there, Lossy compression commonly used for the audio and the video where certain amount of information is lost which is not examined by most of the users.

Keywords: Compression Software, Optimized compression, Optimized Decompression , Huffman method , Lempel Ziv, Lempel Markov algorithms

I. INTRODUCTION

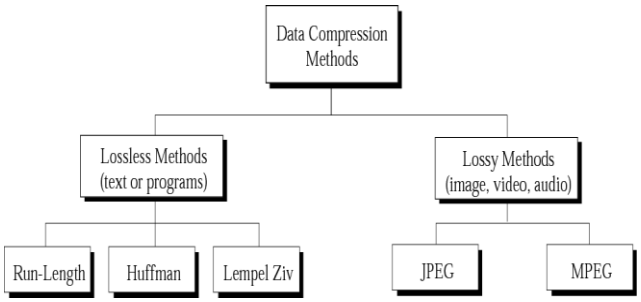
The Data compression is the technique to decrease the size of the data by decreasing the bits [1] in a frame but the sense of the data will not change , and this causes many benefits ,it diminishes space to store the data , time to transfer the data and of course the price . Technically we can say that it is a method to classify the duplicity and to eliminate it .There are two significant ways used for the development of this data compression and

They are (i) Lossless Data Compression (ii) Lossy Data Compression. In Lossless data compression the data is commonly diminishes but its integrity persist the same that is after compression nothing change but only drop in the size [2]. Lossless data compression is used in text file, database tables and in medical image because of law of regulations. In case of Lossy Compression data is reducing by eliminating certain amount of information that causes redundancy, when data is uncompressed a part of the data always still there [3], Lossy compression is used where perfect consistency of the original data is not required. Example of Lossy Compression is the compression of the video and picture data. Some of the main techniques of data compression are Huffman coding, Run Length coding, Lempel Ziv, Arithmetic Coding Dictionary based coding. In this research paper we compare three procedures of the data compression that are (1) Huffman Coding (2) Lempel–Ziv-Welch (3) Lempel-Ziv-Markov algorithms.

II. METHODS AND MATERIAL

A. Compression Methods used by Software

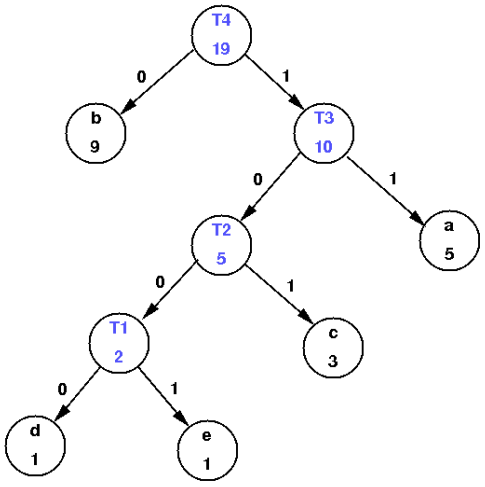
Data compression in the software fundamentally distributed into two parts (1) Lossless data compression and (2) Lossy data compression and these two are sectioned into other fragments as shown below



The Lossless compression commonly deals with the compression of the data [2] , where there is a decrease in the extent of the bits but the original piece of data remains unaffected , It is further divided to other fragments they are (1)Run *Length coding (2) Huffman coding (3) Lempel-Ziv-Welch* (4) Lempel-Ziv-Markov [4]. The Lossy compression method commonly deals with the compression when perfect uniformity with the original data is not required. The lossy compression can be used for the image, audio and video compression, we are going to discuss the ways of lossless data compression.

Compression by Huffman Coding in Software

The simple idea of the Huffman coding is to allocate short code words to those of input blocks with high probabilities and long code words with low priorities [5], this Huffman coding is relatively similar to that of Morse coding. This Huffman code is planned by merging together the two least probable characters and repeating this process till we get only the single character [6]. A code tree is thus generated and Huffman code is obtained by labeling the code tree. For an example below



The benefit of the Huffman coding is that there is no code in the begin of another code .There is no vagueness in the code .The receiver can decode the data without any vagueness that is why sometimes this is also called as instantaneous code.

Huffman coding

Step1: choose two letters x, y from alphabet A with the smallest frequencies and generate a sub tree that has these two characters leaves. (greedy idea) Label the root of this subtree as z.
Step2: set frequency **f(z)=f(x)+f(y).**
Remove x,y and add z creating new alphabet A'=AU{z}-(x,y}.
repeat this iteration with new alphabet A' until an alphabet with only one symbol is left.

Huffman Codes

1. Take the characters and their frequencies, and sort this list by growing frequency
2. All the characters are vertices of the tree
3. Take the first 2 vertices from the list and make them children of a vertex having the sum of their frequencies
4. Add the new vertex into the sorted list of vertices waiting to be put into the tree
5. If there are at least 2 vertices in the list, go to step 3.
6. Read the Huffman code from the tree
7. Decrease size of data by 20%-90% in general
8. If no characters occur more often than others, then no advantage over ASCII

Lempel-Ziv-Welch Encoding Compression in Software

LZ encoding is an example of class of algorithms called dictionary based coding [8]. The idea is to create a dictionary (table) used during the communication session. This compression algorithm extracts the smallest substring that cannot be found in the dictionary. LZW algorithms does not require advance knowledge .it is used for general purpose data compression because of its simplicity and versatility.it is used for the pc utilities that require space double of your double storage of the hard drive. LZW algorithm uses a code table with 4096 as common choice for most of the entries [9]. When coding begins the code table contains only the first 256 entries, Compression is achieved by using codes 256 through 4095 to represent the sequence of bytes .As the coding continues, LZW identifies repeated sequence in the data and adds them in the code table. It is used in UNIX *compress* -- 1D token stream (similar to below). It used in GIF compression -- 2D window tokens (treat image as with Huffman Coding Above). The LZW Compression Algorithm can summarize as follows:

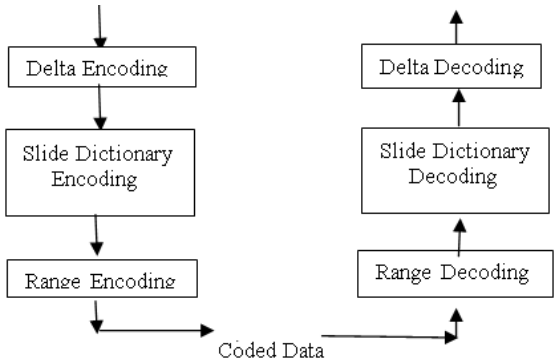
```
w = NIL;

  while (read a character k )
    {
     if wk exists in the dictionary
      w = wk;
     else
       add wk to the dictionary;
       output the code for w;
       w = k;
    }
```

Lempel-Ziv-Markov Used In Software

Lempel-Ziv-Markov Chain Algorithm [10] used for lossless data compression .this algorithm generally use dictionary compression system slightly similar to the LZ77 scheme which was published by Abraham Lempel and Jacob Ziv. It usually features high compression ratio and a variable compression dictionary size .There are numerous points on which LZW classifies these are as follows ,It has generally high compression ratio , its dictionary size variable which is up to 4 gb. Its compressing speed is generally 1Mb/s on 2GHz CPU while it has decompressing speed of 10-20 Mb/s on 2GHz CPU. It generally requires small memory necessities for the decompression (depend on the size of the dictionary).LZMA usually uses a dictionary based algorithm whose output is then encoded with a range encoder [11].It generally uses a composite model to make probability prediction of each bit. Before LZMA most encoder model generally purely byte based, the benefit of the LZMA is that instead of using byte based model, LZMA model using perspective specific to the bit fields in each representation of a literals or phrase [12]. This is as simple as the of generic byte based model but it gives much better compression because it usually escapes collaborating of unconnected bits together, The reason dictionary size is much larger so larger number of memory available to systems.

In LZMA the compressed stream is a stream of bits which is normally encoded using an adaptive series coder. The stream is distributed into packets each packet either labeling a single byte or a LZ77 sequence with its length and distance implicitly or explicitly encoded Data to be coded Decoded Data.

LZMA Coding Scheme



B. Decomposition By Lzma

LZMA data is the minimum level data for decomposition [13] which decodes the data one bit at a time by the range decoder. Context-Based range decoding is invoked by the LZMA algorithms passing it as a reference to the context consist of the unsigned 11bit variable analysis (typically implemented using a 16-bit data type) signifying the forecast the probability of the bit being 1, which is read and updated by the range decoder (and should be initialized to 2^10, representing 0.5 probability).Fixed probability range decoding instead assumes a 0.5 probability, but functions somewhat differently from context-based

range decoding [14]. The range decoder state entails of two unsigned 32-bit variables, *range* (representing the range size), and *code* (representing the encoded point within the range).Initialization of the range decoder entails of setting *range* to 2^32 - 1, and *code* to the 32-bit value starting at the second byte in the stream interpreted as big-endian; the first byte in the stream is completely unnoticed.

Decoding proceeds in this way:

1. Shift both *range* and *code* left by 8 bits
2. Read a byte from the compressed stream
3. Set the least significant 8 bits of *code* to the byte value read

Context-based range decoding of a bit using the *prob* probability variable proceeds in this way:

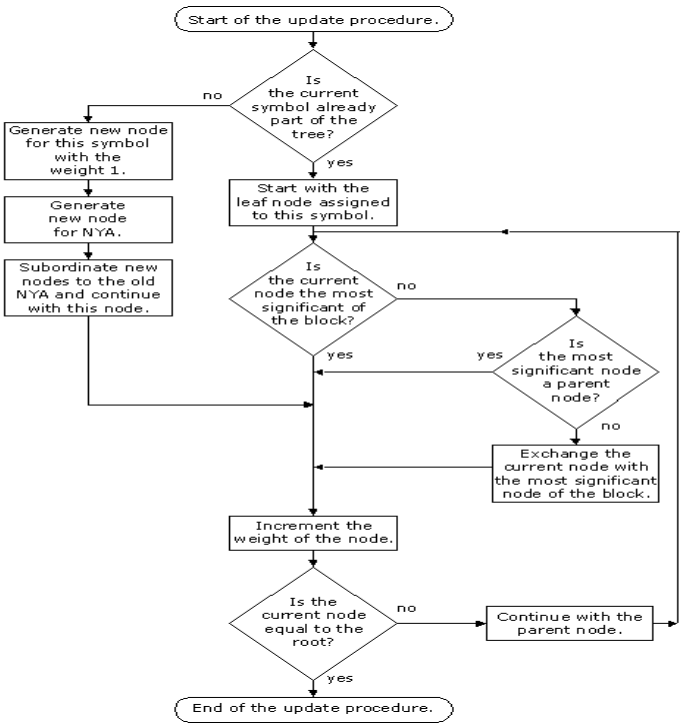
1. If *range* is less than 2^24, perform normalization
2. Set *bound* to floor(*range* / 2^11) * *prob*
3. **If *code* is less than *bound*:**
    1. Set *range* to *bound*
    2. Set *prob* to *prob* + floor((2^11 - *prob*) / 2^5)
    3. Return bit 0
4. **Otherwise (if *code* is greater than or equal to the *bound*):**
    1. Set *range* to *range* - *bound*
    2. Set *code* to *code* - *bound*
    3. Set *prob* to *prob* - floor(*prob* / 2^5)
    4. Return bit 1

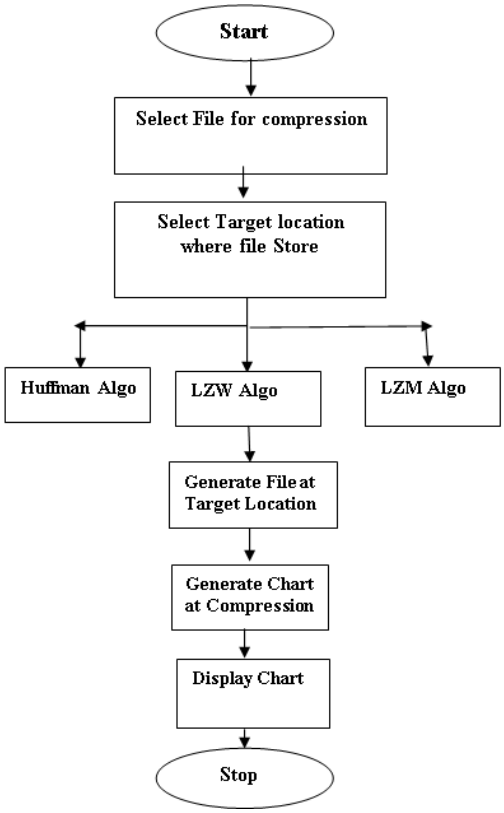
Fixed-probability range decoding of a bit proceeds in this way:

1. If *range* is less than 2^24, perform normalization
2. Set *range* to floor(*range* / 2)
3. If *code* is less than *range*:
    1. Return bit 0
4. Otherwise (if *code* is greater or equal than *range*):
    1. Set *code* to *code* - *range*
    2. Return bit 1

The Linux kernel execution of fixed-probability decoding in redirect, for performance reasons, doesn't contain a provisional branch, but instead deducts *range* from *code* unconditionally, and practises the resulting sign bit to both elect the bit to return, and to produce a mask that is joined with *code* and added to *range*. The division by 2^11 when calculating *bound* and floor process is done before the multiplication, not afterward (apparently to avoid needful fast hardware support for 32-bit multiplication with a 64-bit result) Static probability decoding is not firmly alike to context-based range decoding with any *prob* value, due to the fact that context-based range decoding rejects the lower 11 bits of *range* previously multiplying by *prob* as just defined, while fixed probability decoding only rejects the last bit

Flowchart of Lempel–Ziv–Markov chain algorithm:





III. CONCLUSION

The Huffman encoder accounts an alphabet or symbol to a binary code. The binary code is a mixture of activities of binary bits of varied dimensions. The iteratively performing alphabet will be symbolized by smaller sized binary bits compared with the rarely appearing one (Gonzalez and Woods, 2008). Different from Huffman coding, the LZW coding sets permanent-length code words to variable length order of source symbols (Kelly, 2007). LZW forms a 'dictionary' that embraces words or parts of words of data. When the data wants to be extended, it wants to mention to the dictionary, which in turn shows the LZW code for that word. For double compression, the combination of Huffman followed by LZW (HLZ) and LZW followed by Huffman (LZH) were used. Double compression is inspected in this work to measure that performance when compressing different groups of data. LZMA encoders can easily select which match to output, or whether to disregard the existence of ties and output literals anyway. The ability to recall the 4 most recently used distances means that, in principle, using a match with a distance that will be needed again later may be globally optimal even if it is not locally optimal, and as a result of this, optimal LZMA compression possibly needs information of the complete input and might want procedures too slow to be usable in practice. We can evaluate the compression systems by taking both of the schemes for compression and check which technique will generate the best result. The following graph precisely showed the way to judge the best scheme among the three schemes.

IV. REFERENCES

[1] Introduction to Data Compression, Khalid Sayood, Ed Fox (Editor), March 2000.

[2] Burrows M., and Wheeler, D. J. 1994. A Block-Sorting Lossless Data Compression Algorithm. SRC Research Report 124, Digital Systems Research Center.

[3] Dr. V. Radha and Pushpalakshmi. "Performance Analysis of Lossy Compression Algorithms for Medical Images", Journal of Global Research in Computer Science, Vol. 1, No. 4, Pp 46-50, 2010.

[4] James A. Storer, ―Data Compression methods and theory‖ Computer Science Press, 1988

[5] http://www.cstutoringcenter.com/tutorials/algorithms/huffman.php

[6] http://michael.dipperstein.com/huffma n/index.html

[7] http://en.wikipedia.org/wiki/Huffman_coding

[8] ZIV, J. AND LEMPEL, A. 1978. "Compression of individual sequences via variable-rate coding". IEEE Trans. Inform. Theory 24, 5, 530–536

[9] Hidetoshi Yokoo - "Improved Variations Relating the Ziv – Lempel and Welch-Type Algorithms for Sequential Data Compression" IEEE Transactions on Information Theory.

[10] http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Markov_chain_algorithm

[11] Zongjie Tu and Shiyong Zhang. A Novel Implementation of JPEG 2000 Lossless Coding Based on LZMA.

[12] http://stackoverflow.com/questions/17523458/how-to-create-zip-with-lzma-compression

[13] http://stackoverflow.com/questions/12121062/how-to-decompress-a-lzma-compressed-file-using-bytearray-method-in-as3

[14] https://helpx.adobe.com/flash-player/kb/exception-thrown-you-decompress-lzma-compressed.html