International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Advanced Data Compression Using J-bit Algorithm

Sarita Ambadekar[1], Krushab Gandhi[2], Jay Nagaria[3], Rishabh Shah[4]

Department of Computer Engineering, KJSIEIT, Ayurvihar Complex, Everard Nagar, Sion, Mumbai 400022
Maharashtra, India

Abstract: ***People tend to store a lot of files inside theirs storage. When the storage nears it limit, they then try to reduce those files size to minimum by using data compression software. In this report we discuss a new algorithm for data compression, called j-bit encoding (JBE). This algorithm will manipulates each bit of data inside file to minimize the size without losing any data after decoding which is classified to lossless compression .The performance of this algorithm is measured by comparing combination of different data compression algorithms.***

Keywords: Data Compression, Lossy Compression, Lossless Compression, J-bit Algorithm

1. Introduction

A compression problem involves finding an efficient algorithm to remove various redundancies from a certain type of data. The general question to ask here would be, for example, given a string s, what is the alternative sequence of symbols which takes less storage space? The solutions to the compression problems would then be the compression algorithms that will derive an alternative sequence of symbols which contains fewer number of bits in total, plus the decompression algorithms to recover the original string. How many fewer bits? It would depend on the algorithms but it would also depend on how much the redundancy can be extracted from the original data. Different data may require different techniques to identify the redundancy and to remove the redundancy in the data. There is no 'one size fits all' solution for data compression problems. This gives rise to a variety of data modeling and representation techniques, which are at the heart of compression techniques Morse code, invented in 1838 for use in telegraphy, is an early example of data compression based on using shorter code words for letters such as "e" and "t" that are more common in English. Modern work on data compression began in the late 1940s with the development of information theory. In 1949 Claude Shannon and Robert Fano devised a systematic way to assign code words based on probabilities of blocks. An optimal method for doing this was then found by David Huffman in 1951.In the mid-1970s, the idea emerged of dynamically updating code words for Huffman encoding, based on the actual data encountered. And in the late 1970s, with online storage of text files becoming common, software compression programs began to be developed, almost all based on adaptive Huffman coding. In 1977 Abraham Lempel and Jacob Ziv suggested the basic idea of pointer-based encoding. In the mid-1980s, following work by Terry Welch, the so-called LZW algorithm rapidly became the method of choice for most general-purpose compression systems. It was used in programs such as PKZIP, as well as in hardware devices such as modems. Current image compression standards include: FAX CCITT 3 (run-length encoding, with code words determined by Huffman coding from a definite distribution of run lengths); GIF (LZW); JPEG (lossy discrete cosine transform, then Huffman or arithmetic coding); BMP (run-length encoding, etc.); TIFF (FAX, JPEG, GIF, etc.). Typical compression ratios currently achieved for text are around 3:1, for line diagrams and text images around 3:1, and for photographic images around 2:1 lossless, and 20:1 lossy.

In today's world Data Compression is indeed very much needed. Data compression is used to reduce the size of the file without altering its content, so that maximum content can be stored in a compact space. Existing algorithms gives compression up to some extent, but that's not sufficient. Main objective of this project is to make an algorithm, which when used with existing algorithms, will increase the overall compression ratio.

2. Review of Literature

Lossless data compression is used to compact files or data into a smaller form. It is often used to package up software before it is sent over the Internet or downloaded from a website to reduce the amount of time and bandwidth required to transmit the data [5]. Lossless data compression has the constraint that when data is uncompressed, it must be identical to the original data that was compressed [1]. Graphics, audio, and video compression such as JPG, MP3, and MPEG on the other hand use lossy compression schemes which throw away some of the original data to compress the files even further [4]. We will be focusing on the lossless kind. There are generally two classes of lossless compressors: dictionary compressors and statistical compressors. Dictionary compressors (such as Lempel-Ziv based algorithms) build dictionaries of strings and replace entire groups of symbols [2]. The statistical compressors develop models of the statistics of the input data and use those models to control the final output [3].

There are some well-known data compression algorithms. In this paper we will take a look on various data compression algorithms that can be used in combination with our proposed algorithms [1]. Those algorithms can be classified into transformation and compression algorithms. Transformation algorithm does not compress data but rearrange or change data to optimize input for the next sequence of transformation or compression algorithm[2]

Some of the important algorithms are as follows:

Run-length encoding

Run-length encoding (RLE) is one of basic technique for data compression. The idea behind this approach is this: If a data item *d* occurs *n* consecutive times in the input stream, replace the n occurrences with the single pair *nd* [2]. RLE is mainly used to compress runs of the same byte [2]. This approach is useful when repetition often occurs inside data. That is why RLE is one good choice to compress a bitmap image especially the low bit one, example 8 bit bitmap image.

Huffman Encoding

The Huffman encoding algorithm is an optimal compression algorithm when only the frequency of individual letters are used to compress the data. (There are better algorithms that can use more structure of the file than just letter frequencies.) The idea behind the algorithm is that if you have some letters that are more frequent than others, it makes sense to use fewer bits to encode those letters than to encode the less frequent letters. For instance, take the following phrase: "ADA ATE APPLE". There are 4 As, 1 D, 1 T, 2 Es, 2 Ps, 1 L, and 2 spaces. There are a few ways that might be reasonable ways of encoding this phrase using letter frequencies. First, notice that there are only a very few letters that show up here. It would be silly to use chars, with eights bits apiece, to encode each character of the string. In fact, given that there are only seven characters, we could get away with using three bits for each character! If we decided to simply take that route, that would require using 14 * 3 bits, for a total of 42 bits (and some extra padding for the sake of having correct bit-alignment since you have to use an entire byte). That's not too bad! It's a lot better than the 8 * 14 = 112 bits that would otherwise be required.

But we can do even better if we consider that if one character shows up many times, and several characters show up only a few times, then using one bit to encode one character and many bits to encode another might actually be useful if the character that uses many bits only shows up a small number of times!

The Lempel Zev Welch Algorithm

Dictionary based compression algorithms are based on a dictionary instead of a statistical model [5]. 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.

Arithmetic coding

In arithmetic coding, a message is encoded as a real number in an interval from one to zero. Arithmetic coding typically has a better compression ratio than Huffman coding, as it produces a single symbol rather than several seperate codewords. Arithmetic coding is a lossless coding technique. There are a few disadvantages of arithmetic coding. One is that the whole codeword must be received to start decoding the symbols, and if there is a corrupt bit in the codeword, the entire message could become corrupt. Another is that there is a limit to the precision of the number which can be encoded, thus limiting the number of symbols to encode within a codeword. There also exists many patents upon arithmetic coding, so the use of some of the algorithms also call upon royalty fees. Arithmetic coding, is entropy coder widely used, the only problem is its speed, but compression tends to be better than Huffman (other statistical method algorithm) can achieve [6]. This technique is useful for final sequence of data compression combination algorithm and gives the most for compression ratio

Compression Performances Measuring

Depending on the nature of the application there are various criteria to measure the performance of a compression algorithm. When measuring the performance the main concern would be the space efficiency.The time efficiency is another factor. Since the compression behavior depends on the redundancy of symbols in the source file, it is difficulty to measure performance of a compression algorithm in general.

Following are some measurements used to evaluate the performances of lossless algorithms.

Compression Ratio is the ratio between the size of the compressed file and the size of the source file.

Compression Time is the time taken for the compression and decompression should be considered separately. If the compression and decompression times of an algorithm are less or in an acceptable level it implies that the algorithm is acceptable with respective to the time factor.

Comparing the Performance

The performances of the above discussed algorithms vary according to the type of file, content of the file, compression ratio, compression time etc. If an algorithm gives 20% compression for a file, it may only 5% for some other file. So there is no one specific algorithm for all data compression needs. Therefore, all these factors are considered for comparison in order to identify the best solution. An algorithm which gives an acceptable saving percentage within a reasonable time period for most of the files is considered as the best algorithm.

Proposed Algorithm

J-bit encoding (JBE) optimizes input for other algorithm [1].This is done by manipulating bits and give compressed file as an input for the existing algorithms. The main idea of this algorithm is to split the input data into two data where the first data will contain original nonzero byte and the second data will contain bit value explaining position of nonzero and zero bytes [4]. Both data then can be compress

separately with other data compression algorithm to achieve maximum compression ratio.

Step-by-step Compression Process

- Read input per byte, can be all types of file
- Check whether bit is zero or non-zero
- If bit is non-zero copy it in the data I array and write "1" in the temp array.
- If bits are zero simply copy a "0" in temp array.
- Repeat above steps till there are 8 bits in temp array.
- Once filled write the Byte value of those 8 bits in data II array.
- Clear the temp array.
- Repeat above steps till end of file is reached.

Once done our other existing algorithms can be applied on data II and data I(optional) which will further increase the compression ratio. This further compressed output can be stored as required and while retrieving can be opposite process can be applied.

Step-by-Step Decompression Process:

While decompressing first decompress using existing algorithm then apply the decompression process of the j-bit algorithm which is given as follows-

- Read the input file.
- Read data II per bit.
- Determine whether read bit is '0' or '1'.
- If the bit is '1' then read and write data I to output, if the bit is '0' then write zero byte to output.
- Repeat step 2-5 until end of file is reach.

The compression ratio provided depends on the content of the file, type of the file and many other factors. The following table gives the average compression ratio provided by the above algorithms , by taking various types of files of different inputs data and applying the above algorithms.

Input file size: 100 bytes
Output file size (in bytes):

	Without j-bit	With j-bit
RLE	60	54
LZW	80	34
ARI	83	92

As we can see from the above table j-bit gives best output for LZW but the combination of compression algorithms to be used depends on what data the file carries.

3. Conclusion

This topic discusses and confirms a data compression algorithm that can be used to optimize other algorithm. This algorithm gives better compression ratio when inserted between run length encoding and arithmetic coding .Because some files consist of hybrid contents (text, audio, video, binary in one file just like document file), the ability to recognize contents regardless the file type, split it then compresses it separately with appropriate algorithm to the contents is potential for further research in the future to achieve better compression ratio. The next generation campus wide architecture is a conversed architecture; it has capability to provide voice, video, data triple play service over a single conversed infrastructure, which requires a huge amount of bandwidth and high data rate. Using our system it is possible to reduce bandwidth consumption and eventually reduce the cost.

References

[1] I Made Agus, Dwi Suarjaya," A New Algorithm for Data Compression Optimization"(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No.8, 2012
[2] S.Gavaskar, Dr.E.Ramaraj, R.Surendiran, "A Compressed Anti IP Spoofing Mechanism Using Cryptography ," IJCSNS International Journal of Computer Science and Network Security, VOL.12 No.11, November 2012 .
[3] Salomon, D. 2004. Data Compression the Complete References Third Edition. Springer-Verlag New York, Inc.
[4] Campos, A. S. E. Move to Front. Available: http://www.arturocampos.com/ac_mtf.html (last accessed July 2012).
[5] Campos, A. S. E. Run Length Encoding. Available: http://www.arturocampos.com/ac_rle.html (last accessed July 2012).