Volume 7, Issue 3, March 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Arithmetic Algorithm Improvement

Chiranjeev Singh, Jatin Bansal
Division of Computer Science, Netaji Subhas Institute of Technology,
New Delhi, India

DOI: 10.23956/ijarcsse/V7I3/0131

Abstract*— The purpose of this paper is to propose an algorithm which is an improvement over the Arithmetic Encoding. The improved algorithm works only for texts written in English language. The algorithm works in a similar manner as the Arithmetic coding, but here we exploit the property of uppercase and lowercase English characters and use them to compress the data even further. For each set of uppercase and lowercase letters, we check which one of them occurs first, and based upon that we assign the same digit to it and its corresponding uppercase or lowercase letter. The same procedure is repeated for the next twenty-five pairs. This way, in the encoding process , if a text contains a letter which is present in both uppercase and lowercase format, the one which occurs first is chosen and it results in fewer number of unique characters used in the arithmetic encoding process.

Keywords*— Arithmetic Encoding Algorithm, fraction, Lossless Compression Technique, Data Compression

I. INTRODUCTION

Compression is the art of representing essential features, without including background details. It means representing the information in a compressed form rather than its original or uncompressed form. In other words, using 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. If the algorithm used to encrypt works properly, there should be a significant difference between the original file and the compressed file. 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.

Data compression is a common requirement for most of the computerized applications. They compress any form of data into a suitable form which can be stored in a comparatively less space. This not only saves space, but also time and resources which are used to transfer data from one point to another. There are number of data compression algorithms, which are dedicated to compress different data formats. Even for a single data type there are number of different compression algorithms, which use different approaches. This paper examines the lossless data compression algorithm-Arithmetic Encoding and compares its performance with the improved version discussed below. Both of them are examined and implemented to evaluate the performance in compressing text data. The compression factor of the improved version is comparatively better than that of Arithmetic Encoding.

Arithmetic Encoding is a form of entropy coding used in lossless data compression. Lossless compression techniques reconstruct the original data from the compressed file without any loss of data. Thus the information does not change during the compression and decompression processes. Arithmetic coding assigns a sequence of bits to a message, a string of symbols. Arithmetic coding can treat the whole symbols in a list or in a message as one unit. Unlike Huffman coding, Arithmetic coding doesn´t use a discrete number of bits for each character. The number of bits used to encode each symbol varies according to the probability assigned to that symbol. Low probability symbols use many bit, high probability symbols use fewer bits, i.e., frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. The main idea behind Arithmetic coding is to assign each symbol an interval. It represents the current information as a range, defined by two numbers.

The improved algorithm can be used to compress data even further and can be used with the normal arithmetic code, with some minor modifications. Time complexity of the improved version is approximately same as that of the arithmetic encoding, but at the same time compressed data will occupy less space and can be easily transferred.

II. DESCRIPTION

In this method, a code word is not used to represent a symbol of the text. Instead it uses a fraction to represent the entire source message. The occurrence probabilities and the cumulative probabilities of a set of symbols in the source message are taken into account. The cumulative probability range is used in both compression and decompression processes. In the encoding process, the cumulative probabilities are calculated and the range is created in the beginning. While reading the source character by character, the corresponding range of the character within the cumulative probability range is selected. Then the selected range is divided into sub parts according to the probabilities of the alphabet. Then the next character is read and the corresponding sub range is selected. In this way, characters are read

repeatedly until the end of the message is encountered. Finally a number should be taken from the final sub range as the output of the encoding process. This will be a fraction in that sub range. Therefore, the entire source message can be represented using a fraction. To decode the encoded message, the number of characters of the source message and the probability/frequency distribution are needed.

III. IMPROVEMENT ALGORITHM ANALYSIS

A. Improvement Algorithm Description

The improvement algorithm is similar to the Arithmetic encoding. In the arithmetic encoding, each unique character/symbol is assigned a different number, i.e. , in the fractional part of the range which is taken into account. In this process, the uppercase and lowercase symbols of any particular alphabet are assigned different numbers. This property of English alphabets being represented as uppercase and lowercase symbols is exploited to get better results.

For each particular alphabet, its first occurrence is recorded, i.e., it is kept in record that whether the first occurrence is uppercase or lowercase. The same procedure is repeated for the rest of twenty-five characters. If, for instance, the first character encountered for a particular alphabet is uppercase, then the uppercase letters of that particular alphabet are assigned same numbers as done in the normal Arithmetic encoding, but when its corresponding lowercase letters are encountered, they are not assigned any different number, instead they are also assigned the same number as assigned to the uppercase letter, but in addition an account of the position of occurrence of those particular lowercase letters for that particular alphabet is kept. Those positions can be stored in a vector. This means, positions of either lowercase or uppercase letters of a particular alphabet are stored in the vector, which depends on the first symbol encountered. This is vice-versa, i.e., if the first encountered character is lowercase, same procedure is repeated for uppercase letters. This process of storing the locations of a particular symbol in accordance with the occurrence of first encountered symbol is repeated for all twenty-six alphabets. Twenty-six vectors are maintained to keep track of the lowercase or uppercase letters of each alphabet. This way, lesser number of unique symbols are used in the fractional form of the representation of data and hence, lower is the base of the decimal representation and eventually, lower number of bits are used in the data encoding.

Data decoding is done in a similar manner as done in Arithmetic encoding. The difference is that in the decoding process, we keep track of the pointer location. For instance, for a particular alphabet, if the first encountered character is uppercase, then during the decoding process, if the uppercase letter of that alphabet is encountered in the encoded file, then its particular position vector is checked and if that position is stored in the vector, it is converted to its corresponding lowercase, otherwise it is kept as it is.

B. Algorithm

The input file is read. Twenty six vectors are declared. A map keeps track of characters being visited. The map considers uppercase and lowercase letters as same. It can be said that each element of the map has two keys, an uppercase and lowercase of corresponding letters. Each element of the map contains the first character encountered, either uppercase or lowercase. So in all, the map contains twenty-six elements, each with a single value which can be accessed by two keys.

N^{TH} vector out of 26 vectors is represented as $VECTOR_N$.

PTR is the pointer used to read the text file.

During the traversal of the input file, first encountered character of each particular alphabet is stored in the map.

```
IF (*PTR EQUALS MAP [*PTR])
    PRINT *PTR

ELSE
    IF (*PTR IS LOWERCASE)
        PTR ← VECTOR_N [I]
        PRINT UPPERCASE
        I++

    ELSE
        PTR ← VECTOR_N [I]
        PRINT UPPERCASE
        I++

PTR++
```

After applying this Encoding algorithm, the normal Arithmetic compression can be performed on the modified data. Decoding can be done in a similar manner as done in Arithmetic encoding, the additional part in decoding being to keep track of the positions stored in each of the vectors and modify the content if same position is stored in the vector.

C. Improvement Graph

Here, blue line represents the Arithmetic Encoding while orange line represents the Improvement algorithm. All the numerical values are represented in percentage. This graph shows percentage improvement of the improved algorithm in comparison to Arithmetic Encoding.

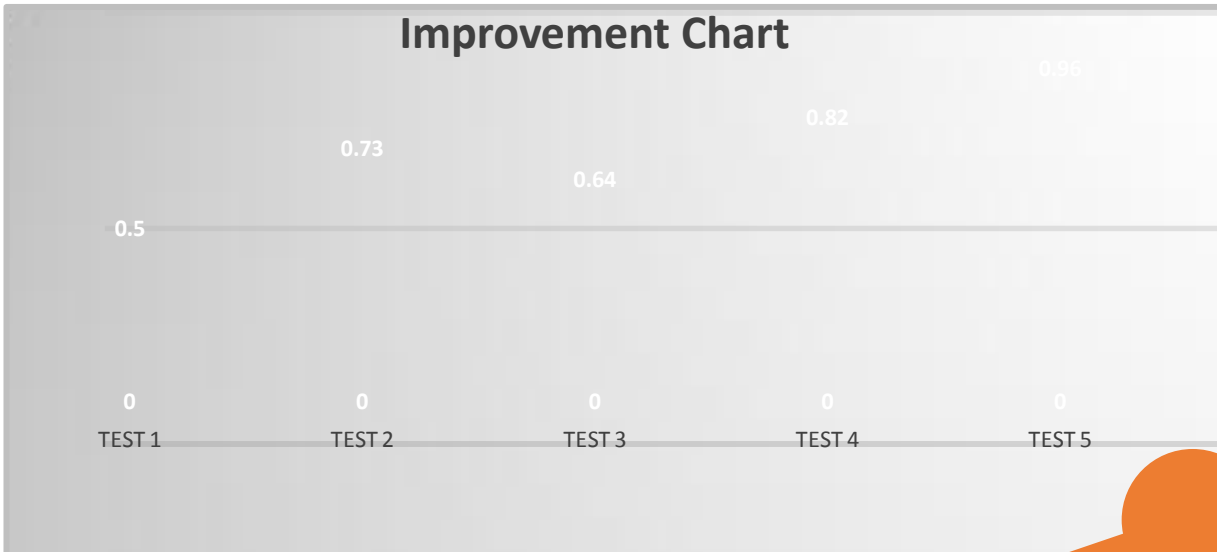

Fig. 1 Improvement Graph

D. Test Case References

i. http://gutenberg.net.au/ebooks/m00017.txt
ii. http://gutenberg.net.au/ebooks04/0400351.txt
iii. http://gutenberg.net.au/ebooks06/0600181.txt
iv. http://gutenberg.net.au/ebooks04/0400291.txt
v. http://gutenberg.net.au/ebooks14/1401701.txt

IV. CONCLUSION

Data compression is considered as one of the most important techniques in the present world. There are many ways to compress data and make it a bit more portable, and one of the ways is discussed in this paper. The above algorithm not only posed as an improvement over Arithmetic encoding, but has also paved a way for various other improvements in the field of data compression. As the technology is advancing, so are the ways to handle it.

REFERENCES

[1] Handbook of Data Compression by David Salomon and Giovanni Motta.
[2] Introduction to Data Compression by Khalid Sayood.
[3] Data Compression with Arithmetic Coding by Mark Nelson
[4] Compression Algorithms by Peter Wayner
[5] A Concise Introduction to Data Compression by David Salomon
[6] Fundamental Data Compression by Ida Mengyi Pu

