Journal of Scientific Research & Reports
4(1): 60-67, 2015; Article no.JSRR.2015.009
ISSN: 2320–0227

SCIENCEDOMAIN *international*
www.sciencedomain.org

SCIENCEDOMAIN

An Efficient Compression Technique Using Arithmetic Coding

Ahsan Habib[1*] and Debojyoty Chowdhury[1]

[1]*Department of Computer Science and Engineering, Shahjalal University of Science and Technology, Sylhet, Bangladesh.*

This work was carried out in collaboration between both authors. Author AH designed the study, wrote the protocol, wrote the first draft of the manuscript and managed the experimental process. Author DC managed the literature searches; analyses of the study performed the compression analysis. Both authors read and approved the final manuscript.

Article Information

DOI: 10.9734/JSRR/2015/12846
Editor(s):
(1) Luigi Rodino, Faculty of Mathematical Analysis, Dipartimento di Matematica, Università di Torino, Italy.
Reviewers:
(1) H D Arora, Department of Mathematics, Amity Institute of Applied Sciences, Amity University, Noida – 201303, India.
(2) Garba S. Adamu, Department of Mathematics and Computer, Waziri Umaru Federal Polytechnic, Birnin Kebbi, Nigeria.
(3) Anonymous, Near East University, Cyprus.
(4) Anonymous, Technical College – Kirkuk, Iraq.
(5) Anonymous, Hanoi University of Science and Technology, Vietnam.
Complete Peer review History: http://www.sciencedomain.org/review-history.php?iid=689&id=22&aid=6327

Original Research Article

Received 21st July 2014
Accepted 16th September 2014
Published 5th October 2014

ABSTRACT

Data compression is a special kind of technique for representing various types of data such as text, audio, video, images etc. Data compression is used to reduce the number of bits and to store data. Different types of data compression mechanism are widely being used for compressing several types of data. In the field of data compression Arithmetic Coding is one of the most important technique which plays a vital role in lossless data compression. The main objective of this paper is to develop an efficient encoding algorithm using Arithmetic Coding. This paper has some test cases and found that the proposed technique is more efficient and effective than binary Huffman coding in terms of both speed and memory usage.

Keywords: Arithmetic coding; huffman coding; text compression; data compression; encoding.

1. INTRODUCTION

Compression is one kind of process or technique which has been used to reduce the amount of data needed for storage or transmission of data like text, audio, video, and image etc. Data compression is used in a computer system to store data that occupies less space than the original form [1].

**Corresponding author: E-mail: ahferoz@gmail.com;*

There are two types of data compression process in a computer system. One is lossless and another one is lossy technique. In lossless data compression there is no loss in the information, and the data can be reconstructed exactly same as the original. Lossless compression is mostly used in text files. In lossy compression some data in output is lost but not detected by users. Lossy compression mostly used in audio, video and image files [2]. In modern time several types of data compression techniques are used such as Huffman, Arithmetic coding, LZW, Shannon-Fano, Run length coding etc.

Among them Huffman and Arithmetic coding are mostly used for text compression. In the research [3], it shows the comparison between Huffman and Arithmetic coding by using some data. Huffman coding is an entropy encoding algorithm used for lossless data compression in computer science and information theory. Huffman coding based on the frequency of a data item [4]. Huffman coding is most commonly used technique in data compression. Some people use it as one step in a multistep compression technique [5]. In Arithmetic coding, a message is represented by an interval of real numbers between 0 and 1. The performance of Arithmetic coding is optimal without the need for blocking of input data [6].

Arithmetic coding is a data compression technique that encodes data (the data string) by creating a code string which represents a fractional value and differs considerably from the more familiar compression coding techniques, such as prefix (Huffman) codes. Also, it should not be confused with error control coding [7].

In compression when we consider all different entropy-coding methods and their possible applications, arithmetic coding stands out in terms of elegance, effectiveness and versatility. Since it is able to work most efficiently in the largest number of circumstances and purposes [8]. Arithmetic Coding is superior in most respects to the better-known Huffman [9] method. It represents information at least as compactly sometimes considerably more so [6].

Flexibility is most important advantage of Arithmetic Coding. This advantage is significant because large compression can be obtained only through the use of sophisticated models of the input data [10].

2. ARCHITECTURE

2.1 Huffman Coding

Huffman is a popular method of data compression. In computer science Huffman coding based on entropy encoding algorithm used in lossless data compression. The Huffman coding creates a variable-length codes that are integral number of bits. It is not very different from Shannon-Fano algorithm. The inventor of Huffman coding algorithm is David Huffman, who invented this technique whenever he was a Ph.D. student at MIT [8]. Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code (sometimes called "prefix-free codes", that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol) that expresses the most common source symbols using shorter strings of bits than are used for less common source symbols. Huffman was able to design the most efficient compression method [11]. The statistical model of Huffman coding is shown in the Fig. 1.

In the case of Huffman coding, the actual output of the encoder is determined by a set of probabilities. When using this type of coding, a symbol that has a very high probability of occurrence generates a code with very few bits. A symbol with a low probability generates a code with a larger number of bits.

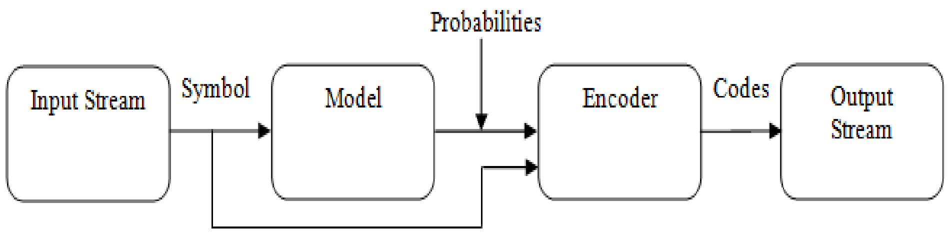

Fig. 1. Statistical model of Huffman coding

A Huffman code [12] is generally done by following a binary decoder tree.

i. First start with a list free node, where each node corresponds to a symbol in the alphabet.
ii. Select two free leaf nodes with the lowest weight from the list.
iii. Construct a parent node for these two leaf nodes, the weight of parent node is equal to the sum of two child nodes.
iv. Remove the two child nodes from the list and the parent node is added to the list of free nodes.
v. Repeat the process starting from step-2 until only a single tree remains.

In Fig. 2, the binary Huffman tree processes the substring “BOOMBOOM”. In Huffman encoding the substring “BOOMBOOM” generate a bit string 001101001101 that occupy 12 bits. The Huffman generated code-word for some test cases are shown in Table 1.

In data structure requires O (log n) time per insertion, and a tree which have n leaves has 2n-1 nodes, this algorithm operates in time O (n log n).

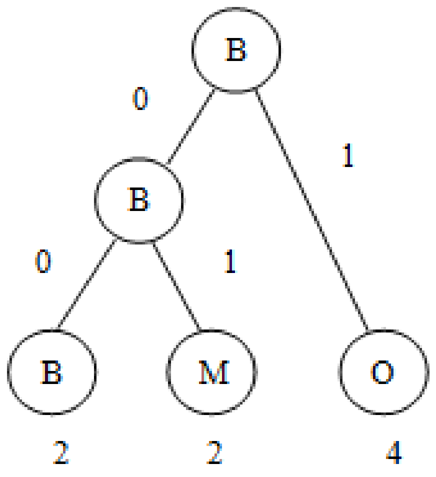

Fig. 2. Huffman binary tree

Table 1. Find bits in huffman encoding

Message	Huffman Coding	Bits
Cocacola	1011001101000001	16
Boomboom	001101001101	12
Data	001011	6
Book	001101	6
Lossless	01000110100111	14
Be a bee	0110010000010111	16

2.2 Arithmetic Coding

Arithmetic coding is important because it was invented as an alternative of Huffman coding after a gap of 25 years. It is superior in performance to the Huffman coding when the substring is fairly small or redundancy in a alphabet. In 1948 this idea first comes from Shannon’s observation that message may be encoded by their cumulative probability. This can also be seen in static Huffman coding where a sequence of symbols is assigned on Huffman code-word to achieve a better compression ratio. Arithmetic coding is now the method of choice for adaptive coding on multilevel symbol alphabets because of:

- Its speed.
- Minimum storage requirements.
- Effectiveness of compression.

In Arithmetic coding successive symbols of the text reduce the size of the interval. Likely symbols reduce the range by less than the unlike symbols and added fewer bits to the message [6]. The range of the interval (0,1] denoting it half open interval. Starting with the interval (0,1] then every interval is divided in several subintervals. The size of the interval is proportional to the current probability of the corresponding symbol of the alphabet. The subinterval from the coded symbol is then taken as the interval of the next symbol. Approximately the codeword length is equal to -$\log_2 p(s)$, where p(s) is the probability of the source sequences.

The basic idea of Arithmetic coding is to assign short codeword to more probable events and longer codeword to less probable events. Arithmetic coding provides a special mechanism for removing redundancy of the input data.

There is another thing in the Arithmetic coding is that there is no problem which characters are assigned in the interval range. As long as it is done in the same manner by both the encoder and the decoder. The three symbol set used here in the message “DATA” would look like in the Table 2.

Once character probabilities are known, individual symbols need to be assigned a range along a “probability line,” nominally 0 to 1. It doesn’t matter which characters are assigned which segment of the range, as long as it is done in the same manner by both the encoder and the decoder.

Here is the original encoding algorithm for arithmetic coding.

```
  Set low to 0.0
  Set high to 1.0

While there are still input symbols do

    get an input symbol
    code_range = high - low.
    high = low + range*high_range(symbol)
    low = low + range*low_range(symbol)
     End of While output low
```

Table 2. Probability analysis using Arithmetic coding

Character	Probability	Range
D	1/4	0.00 – 0.25
A	2/4	0.25 – 0.75
T	1/4	0.75 – 1.00

Following the encode process we can find the message “DATA” as below:

Character	Low value	High Value
D	0.0	1.0
A	0.0	0.25
T	0.0625	0.1875
A	0.15625	0.1875
	0.171875	0.1875

So the final low value, 0.171875 which generates 6 bits (0.001011) of the message.

In Table 3 some substrings are taken and find the number of bits for each of these substrings according to Arithmetic coding.

Table 3. Find bits in arithmetic coding

Message	Arithmetic coding	Bits
Cocacola	001010100100001	15
Boomboom	1000111001	10
Data	10011	5
Book	101	3
Lossless	101100010111	12
Be a bee	011110001001	12

2.3 Experimental Design

The experimental design of the research is shown in Fig. 3. After taking the raw text or paragraph as input, the system will calculate the probability of each symbol. The cumulative probability can be calculated from the initial probability. The proposed formula will be applied in the probability and we will get a decimal number as output. The decimal number will be converted to binary number which is our encoded data. Finally we will compare our performance with most popular Huffman coding.

As we know Arithmetic coding work within an interval of 0 and 1 then subdivide the substring according to this process and find the final bit of input string. We start with a current interval (L,H] initialized (0,1]. For each symbol of the file, we perform two steps that subdivide the current intervals into subintervals, one for each possible alphabet symbol. The size of a symbol subinterval is proportional to the estimated probability that the symbol will be the next symbol in the text, according to the model of the input. Select the subinterval corresponding to the symbol that actually occurs next in the file, and make it the new current interval. The final interval will be that interval which matches with the input text.

Finally the lowest decimal number of the interval will be processed for binary data. Fig. 4 shows the details of the calculations.

In Fig. 4 we look at the frequency for different letters. And generate a string ‘SLOE’ according to their frequency for the message of text ‘LOSSLESS’. We encode the string ‘SLOE’ by dividing up the interval (0,1] and allocate each letter as an interval whose size depends on the probability. Our message of text starts with ‘L’, so we take the ‘L’ interval first and divide it up again in the same way. The interval between LS and LL starts with the lower interval of ‘L’ 4/8. Then the lower interval between LS and LL will be 4/8. Then we find the higher interval by calculating lower interval + (frequency of L / length of text * frequency of S / length of text). So, we find 4/8+(2/8*4/8) = 40/64 as higher interval for LS. Then for the next interval between LL and LO we take the higher interval of the previous one LS as our lower interval. So our lower interval for LL is 40/64. To find the higher interval between LS and LL we use the previous calculation like lower interval + (frequency of L/ length of text * frequency of L/ length of text). By put the value we get the higher interval as 40/64+(2/8 *2/8) = 44/64. In third step we see the next letter of the message of text ‘O’ so now we subdivide the ‘O’ interval in the same way. After continuing this process we get our desired message ‘LOSSLESS’ with lower and higher interval. Thus we can represent the text message ‘LOSSLESS’ with these two intervals.

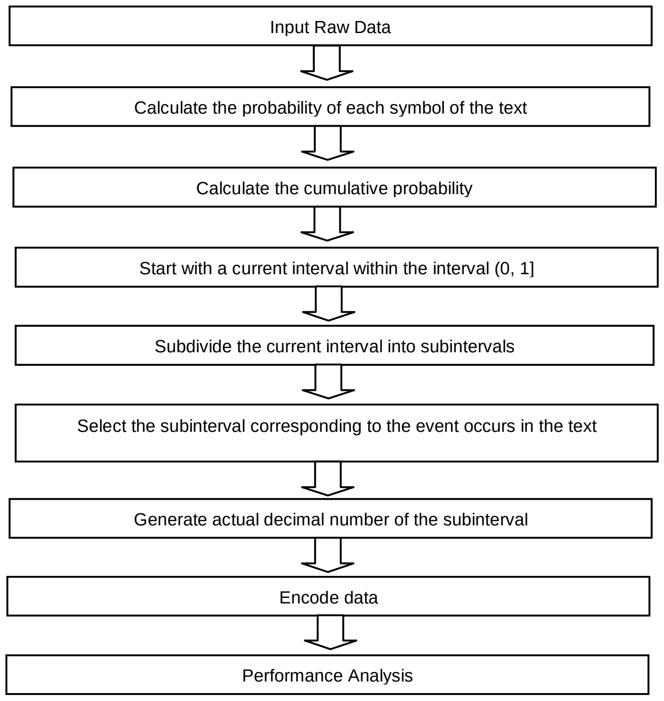
Input Raw Data
Calculate the probability of each symbol of the text
Calculate the cumulative probability
Start with a current interval within the interval (0, 1]
Subdivide the current interval into subintervals
Select the subinterval corresponding to the event occurs in the text
Generate actual decimal number of the subinterval
Encode data
Performance Analysis

Fig. 3. Experimental design of arithmetic encoding

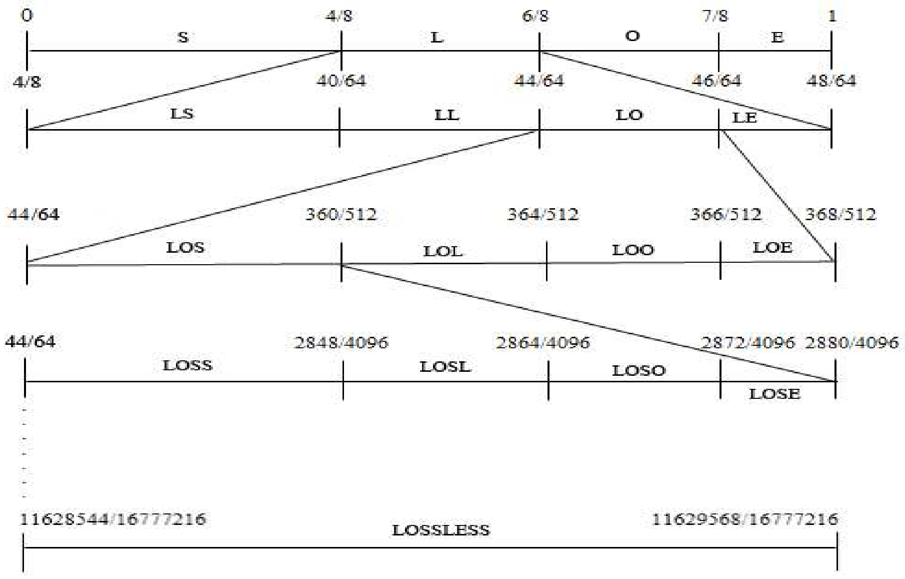

Fig. 4. The process of the proposed technique

Then we find out the binary number of these two intervals and take the smaller one as our output.

Thus, after continuing the technique we represent the message as any number in the interval [11628544/ 16777216, 11629568/ 16777216). Then we convert the two numbers as a binary one to get our result.

$$\frac{11628544}{16777216}$$

$$= \frac{2^{12} * 2839}{2^{24}}$$

= 0.101100010111 (Result)

$$\frac{11629568}{16777216}$$

$$= \frac{2^{10} * 11357}{2^{24}}$$

= 0.10110001011101

In these two intervals we chose the small binary one as our desired result. The details of the proposed technique are explained with the following steps:

STEP 1: We have generated a new string or text according to the frequency of original message. We encode the text by dividing up the interval (0, 1] and allocate each letter or symbol in an interval range whose size depends on the probability.

STEP 2: First of all we took the first symbol of the original message or text as input and thus the length of text will be two. Define new lower interval which is found from step one. Find out its next interval by multiplying according to the frequency of these two symbols separately. Finally add the sum of the above multiplication with the lower interval of the latest segment.

STEP 3: In this step we will take next symbol of the text and its length will be three. Define its new lower interval from step 2. Finally the output will be the lower interval.

STEP 4: However, in the third step the length of text will be four and as usually take the input of a new symbol. Find out the new lower interval for the current segment of the step from step 2. Calculate the higher interval for each segment of the existing text length of four by multiplying all the frequency of each symbol one another and add them with the current lower interval.

STEP 5: Take a new symbol as input and length of text will be increases to five. Then set the new lower interval from step 4 for this step. For every segment of the text find out their next interval again with multiplying the frequencies of all the five symbols individually and finally add them with lower interval. Thus we will find the output of a lower interval.

3. IMPLEMENTATION

3.1 Arithmetic Coding

In the pseudo code we define two variables called low and high for store lower and higher interval accordingly. In line 4 we check the redundancy of the Text symbols with a function redundancy_check () and store it in an array Red_msg[]. Then in line 5 a loop from 1 to length of Text will work while another loop in line 7 from 1 to length of the array Red_msg[] will going on. Then we store the value of j in a variable position in line 8. While in line 9 we check if the value of i is 1and the comparison of encoded string and input Text is not equal i then a function Store_interval () is used in line 11 to store some value. In Store_interval () the parameters low, high, string and position are used to pass several data. Through this function we store the value of lower and higher interval in two variables low, high with the encoded string and position of the value.

In line 14 we checks if the comparison between the Text and string is minimum and equal to i then Store_interval () function is used again to store values finally and break the nested loop on line 7. Thus continuing the process and find the value of string as our input Text. Finally in line 17, 18 we convert the lower and higher interval of the encoded Text with a function DecToBin () to binary one and store them in two separate array of variable called low and high.

3.2 Encoding Algorithm

Encoding ()

BEGIN

```
1   Input: Text
2   Output: Encoded bit stream
3   low = 0, high = 1;
4   Red_msg [] = redundancy_check (Text);
5   For i=1 to length of Text
6   low = high;
7   For j=1 to length of Red_msg []
8   position = j;
9   if (i ==1 AND compare  not equal i)
10  string [] =  Red_msg[j];
11  Store_interval (low, high, string,
                         position);
12  elseif (i>1)
13  string [] = Text [i- 1]*Red_msg [j];
14  If (compare equal to i)
15  Store_interval (low, high, string,
                         position);
```

```
16 break;
17 low [position] = DecToBin (low);
18 high [position] = DecToBin (high);
19 END
Store_interval (low, high, string, position)

{   if(position>1)
            String [position].low=high;
    else
            String [position].low=low;
    low = String [position].low;
    temp = 1;
    for k = 1 to length of string
            val = redundancy_check
    (string[k]) / length of Text * temp;
            temp = val;
    String [position].high= temp;
    high = String [position].high;
    Encoding_string [position] = string; }
```

4. PERFORMANCE ANALYSIS

In this research authors show a new implementation technique of Arithmetic coding. The proposed data structure is array based and simple to search. The proposed data structure does not need any branch or pointer, thus is very compact as compared with the existing schemes. The complexity of the original arithmetic coding is O(n) whereas the complexity of the proposed technique is O(logn). The performance of proposed technique is also better than a popular algorithm for arithmetic coding implementation proposed by [13]. The technique also compared with a contemporary Huffman based compression technique [14] and it is performed better in terms of memory usage.

In these following case examples in Table 4 shown that the Huffman algorithm takes more bits stream than our proposed algorithm using Arithmetic coding. So, we can assure that our proposed technique using Arithmetic coding takes less bits than Huffman coding in text compression.

Table 4. Comparison between bit streams of Huffman and proposed Arithmetic Algorithm

Message	Bits in huffman	Bits in proposed arithmetic coding
Cocacola	16	16
Boomboom	12	11
Data	6	6
Book	6	5
Be a bee	16	14

In the Table 4, it has been shown that the proposed technique always performs better than the Huffman technique for most test cases. Though the technique is compared with some sample data in the above Fig., but the algorithm is applicable for any size of text or paragraph data.

4. CONCLUSION

The main contribution of this research is to develop a new encoding algorithm by using the principle of arithmetic coding. In the research we have also compare the performance of Huffman encoding algorithm with our developed algorithm. We have shown that proposed techniques significantly better than Huffman Coding techniques in terms of memory requirements. The proposed technique is tested for Computer devices, however it may be implementing for mobile devices also. In future an efficient algorithm may be designed for decode the encoded bit stream.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Jacob N, Somvanshi P, Tornekar R. Comparative Analysis of Lossless Text Compression Techniques. International Journal of Computer Applications (0975-8887). 2012;56(3):17-21.
2. Shanmugasundaram S, Lourdusamy R. A Comparative Study of Text Compression Algorithms. International Journal of Wisdom Based Computing. 2011;1(3):68-76.
3. Porwal S, Chaudhary Y, Joshi J, Jain M. Data Compression Methodologies for Lossless Data and Comparison between Algorithms. International Journal of Engineering Science and Innovative Technology (IJESIT). 2013;2:142-147.
4. Sharma M. Compression Using Huffman Coding. IJCSNS International Journal of Computer Science and Network Security. 2010;10:133-141.
5. Srinivasa OR, Setty SP. Comparative Study of Arithmetic and Huffman Data Compression Techniques for Koblitz Curve Cryptography. International Journal of

Computer Applications (0975–8887). 2011;14:45-49.

6. WittenIH, Neal RM, Cleary JG. Arithmetic Coding For Data Compression. Communications of the ACM. 1987;30:520-540.
7. Langdon GG. Jr, Compression Using Huffman Coding. IBM Journal of Research and Development. 1984;28:135-149.
8. Said A. Jr, Introduction to Arithmetic Coding-Theory and Practice. Imaging Systems Laboratory, HP Laboratories Palo Alto, HPL- 2004-76; 2004.
9. Huffman DA. A Method for the Construction of Minimum Redundancy Codes. Proceedings of the I.R.E. 1952;40:1098-1101.
10. Howard PG, Vitter JS. Practical implementations of arithmetic coding. Springer US; 1992:85-112.
11. Huffman coding, Access at Available:http://en.wikipedia.org/wiki/Huffman_coding, last access at; 2013.
12. Suri PR, Goel M. Ternary Tree and Clustering Based Huffman Coding Algorithm. International Journal of Computer Science Issues. 2010;7(5):394-398.
13. Hashempour H. Application of arithmetic coding to compression of VLSI test data. Computers, IEEE Transactions. 2005;54:1166-1177.
14. Habib A, Hoque ASM, Hussain MR. H-HIBASE: Compression Enhancement of HIBASE Technique Using Huffman Coding, JOURNAL OF COMPUTERS. 2013;8(5).