Review on Lossless Compression Techniques

Harika Devi Kotha¹, Madhumitha Tummanapally², Vikash Kumar Upadhyay³

¹Assistant Professor, Department of ECE, ICFAI FOUNDATION FOR HIGHER EDUCATION -FST, Hyderabad, India.
²,³ Students, Department of ECE, ICFAI FOUNDATION FOR HIGHER EDUCATION -FST, Hyderabad, India.

Corresponding Author Email: ecmharika@ifheindia.org.

Abstract: In the present world data compression is used in every field. Through data compression the bits required to represent a message will be reduced. By compressing the given data, we can save the storage capacity, files are transferred at high speed, storage hardware is decreased so that its cost is also decreased, and storage bandwidth is decreased. There are many methods to compress the data. But in this paper, we are discussing about Huffman coding and Arithmetic coding. For various input streams we are comparing adaptive Huffman coding and arithmetic coding and we will observe which technique will be more efficient to compress the data.

Index Terms: Compression, Huffman coding, Arithmetic coding

1. Introduction

Data compression is widely used in all the fields. The main aim of data compression is to minimize the number of bits required code an information or data, which helps minimize the hardware(inventory) required to transfer or store the given data. It encompasses many hardware and software techniques which have only one in common that is compressing the data. Data compression is used in audio compression, video compression, image processing, data transfer, digital telecommunication network [1] [2] etc.

Basically, Data compression methods are broadly classified into two categories that are as follows:

• Lossy compression.
• Lossless compression.

1.1 Lossy compression. Lossy compression as the name says, these methods encounter some loss of information while decompressing the compressed information. These techniques can be applied in graphics, images and digitalized voice. Many of them are adjusted to different level of quality, which gives higher accuracy in transfer of less effective compression.

1.2 Lossless compression. Lossless compression techniques code the data and transfer them accurately; there will not be any kind of loss in data while decompressing the compressed information. This is applied to store database records, spreadsheets, word files etc. In general, data compression is done by taking a stream of symbols and they are transferred into codes. For effective coding the code word should contain fewer bits than original stream of symbols. This decision was made by certain models. One of the methods which is used in data compression is Huffman coding. This paper reviews only lossless compression techniques, such as Adaptive Huffman coding and Arithmetic Coding.
The organization of this paper is as follows: In Section-II the Background of lossless compression techniques is discussed. Section-III provides the algorithms used in Adaptive Huffman coding and Arithmetic coding. Section IV covers the observations. And finally conclusion is presented in section V

2. BACKGROUND OF LOSSLESS COMPRESSION TECHNIQUES

In Huffman coding a symbol which is higher probability of occurrence generates less number of bits; a symbol with less probability of occurrence generates higher number of bits to transfer. There are many methods to acquire Huffman coding but now we will be discussing in brief about adaptive Huffman coding and dictionary coding and compare the bits which are generated by both of them.

Adaptive Huffman coding

Adaptive Huffman coding was first published by Faller (1973) and later Gallager (1978), independently. In 1985 Knuth made a little modification and so the algorithm was FGK [3]. Adaptive Huffman coding is also known as Dynamic Huffman coding, it is a lossless compression. When symbol are being transmitted it permits building the code, having no initial knowledge of source distribution that allows one-pass encoding and adaption to change conditions in data[4]. In this type, the statistics of the source data is not known to the receiver as well as to the transmitter during the beginning of the transmission. Initially the tree contains a single node called as NYT (Not Yet Transmitted), which is associated with weight 0.[5]. During the process the tree gets updated and all the new symbols will be attached by splitting NYT node. The same procedure is followed for both encoding and decoding procedure. The general flow of program using adaptive Huffman coding [4] for encoding is shown in figure.1.

Figure.1 Procedure for adaptive Huffman coding.
Drawbacks of adaptive Huffman coding are: -

1. It is very complex to construct the binary tree which in-turn makes it unsuitable for sensor nodes [6].
2. As it does not know anything about the data at initial stage so it will not work efficiently for compression.

Arithmetic Coding:
Arithmetic coding is a lossless compression process. It generates variable length codes. Generally a string is taken and every letter is coded with a fixed number of bits. In this type of coding less frequently occurred letters/symbols are given with larger bit representation and more frequently occurred symbols/letters are coded with fewer bits as to save the memory. It is useful when we deal with small alphabets with highly skewed probabilities. When we want to keep modeling and coding separately the best way is to compress the data using arithmetic coding. This gives a code word with length which is optimal so it achieves high compression of data.

In this type of coding we will calculate a unique identifier called "tag" for the input sequence to be coded. This identifier will become the binary code for the sequence. This makes the work easier as there is no need to generate code words for all sequences having length 'm' instead we will generate a tag for the input sequence and that is the unique binary code for that sequence.

Tag generation is done in 2 ways, we will divide the data into an interval {0, 1} and then according to their probabilities we generate the tag, next one is we generate the tag using scaling method. However the generation of tag will be the unique code for the input sequence but we haven't given the boundary conditions to the tag so some codes can be infinitely long so that the code is not efficient. To solve this problem we have to generate a truncated binary representation.

![Encoding block diagram of Arithmetic coding.](image-url)
3. ALGORITHMS

3.1. ADAPTIVE HUFFMAN CODING

First time if the symbols occurs then NYT (not yet transmitted) is splitted and if symbol already exists then add to the symbol.

In tree always left side weights < right side weights, if it is not then we have to swap the left child and right child.

We have assign logic 0 to left child of the tree and logic 1 to right child of the tree.

We have to simultaneously code the symbol.

To code the symbol we have to follow the following steps:

- If $1 \leq \text{position of symbol} \leq 2r \quad [r=10, e=4 \text{ for English alphabets}]$ then symbol is encoded as $(e+1)$ bit and binary equivalent of $(k-1)$.
- Else, Position of symbol $(k) \geq 2r$ then symbol is encoded as $e$ bit and binary equivalent of $(k-r-1)$.

If the symbol occurred is already coded then we have to encode the symbol by tracing the path of the symbol.

3.2. Arithmetic coding

First divide the tag interval $\{0, 1\}$ into two parts they are $\{0, 0.5\}$ and $\{0.5, 1\}$.

We consider $\{0, 0.5\}$ as lower half and $\{0.5, 1\}$ as upper half.

Pick first symbol from the input sequence and find upper limit and lower limit of the symbol using the following formula [3]

$$l_n = l(n-1) + (u(n-1) - l(n-1)) f(xn - 1)$$

$$u_n = l(n-1) + (u(n-1) - l(n-1)) f(xn)$$

Where $l^n$ is lower limit, $u^n$ is upper limit and $f(x)$ is probability density function. Then check whether the generated limits are lying in which half, if the limits lie in lower half we will transmit '0', if they lie in upper half then transmit '1'. And then rescale the limits.

If they lie in lower half perform $(2^nA)$ operation, if they lie in upper half then perform $(2(A-0.5))$ operation and then transmit '0' or '1' according to that.

Repeat above two steps till the calculated lower and upper limits fall neither in lower and upper bound

If the upper and lower limits doesn't constrain to either of the lower and upper bound then stop this process and pick the next symbol.

By performing this for all the symbols in the sequence then a unique binary code is generated for the sequence.
4. OBSERVATIONS

The following inputs are considered and we performed adaptive Huffman coding and arithmetic coding on given message bits that are:

1. ACBA
2. ABBDC

| S. No. | Message | Adaptive Huffman coding | Arithmetic coding |
|--------|---------|-------------------------|-------------------|
| 1      | ACBA    | 0000000010000010        | 1100011           |
| 2      | ABBDC   | 000000001110001100010   | 000000011101      |

It is observed that when probability of the symbols is not in the neighbourhood of each other, then arithmetic coding yields less number of bits, so it is preferable.

5. Conclusion

For two specific sequences with random probabilities adaptive Huffman coding requires more number of bits than arithmetic coding. So, arithmetic coding will require less memory to compress the data compared to adaptive Huffman coding. Taking these reasons into consideration arithmetic coding is more preferable than adaptive Huffman coding in data compression techniques.

References

[1] Bei Chen1, Hong-Cai Zhang, Wen-Lun Cao2, Jian-Hu Feng, “Huffman Coding Method Based On Number Character”, "Proceedings Of The Sixth International Conference On Machine Learning And Cybernetics", 2007, 2296-2299.

[2] Subhra J. Sarkar, Ipsita Mondal, Palash K.Kundu. "A novel data compression approach to reduce string size for PLCC based DAS", 2015 Annual IEEE India Conference (INDICON), 2015

[3] The teacher. info, [online], source : http://theteacher.info/index.php/computing-principles-01/1-3exchanging-data/1-3-1-compression-encryption-and-hashing/2184-dictionary-coding-for-losslesscompression.

[4] Jean-loup Gailly, Mark Nelson, “The Data Compression Book", 0001-0012.

[5] Khalid Sayood, et.al., " Introduction to data compression " text book, fourth edition, PP 91-124

[6] Tharini, C., and P. Vanaja Ranjan. "Design of Modified Adaptive Huffman Data Compression Algorithm for Wireless Sensor Network", Journal of Computer Science, 2009.

[7] Xrysovalantis Kavousianos, Emmanouil Kalligeros, Dimitris Nikolos, "Brief Contributions Optimal Selective Huffman coding For Test-data Compression", "IEEE Transactions On Computers" , 2007,56[8], 1146-1152.

[8] H. K. Reghbati, "An Overview of Data Compression Techniques", "IEEE Transactions on Computers", 1981, 0071-0075.

[9] C. Tharini, P. Vanaja Ranjan, “Design of Modified Adaptive Huffman Data Compression Algorithm for Wireless Sensor Network", "Journal of Computer Science", 2009, 5[6], 0466-0470.
[10] Mamta Sharma, "Compression Using Huffman Coding", “International Journal of Computer Science and Network Security”, 2010, 10[5], 0133-0141.

[11] John G. Cleary, Ian H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching", "IEEE Transactions On Communications", 1984,32[4], 0396-0402.

[12] S.R. Kodituwakku, U. S.Amarasinghe, "Comparison of Lossless Data Compression Algorithms for Text Data", "Indian Journal of Computer Science and Engineering", 1[4], 0416-0426.

[13] Chun-Jen Tsai, "Dictionary-based Coding Techniques", 2014, 0001-0031.

[14] Wikipedia contributors. "Adaptive Huffman coding." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 29 Jan. 2018. Web. 23 Oct. 2018.