**Huffman Coding: a Case Study of a Comparison Between Three Different Type Documents**

Konstantinos D. Pechlivanis

**Abstract**—We examine the results of applying Huffman coding to three different style documents: a novel from the 19th century, an HTML document of a modern news website and a C language source code.

**Index Terms** - Data compression, Lossless source coding, Entropy rate of a source, Information theory

I. INTRODUCTION

Ever since the development of electronic means for the transmission and information processing, has emerged the need of reducing the volume of data transmitted.

Data compression theory was formulated by Claude E. Shannon, in his 1948 paper "A Mathematical Theory of Communication" [1]. Shannon proved that there is a fundamental limit to lossless data compression. This limit, called entropy rate, is denoted by H. It is possible to compress the source in a lossless manner, with compression rate close to H and it can be mathematically proven that it is impossible to have a better compression rate than H.

One important method of transmitting messages is to transmit in their place sequences of symbols [2]. For the best performance of communication systems, sought as much as possible compact representation of messages which is achieved by removing the redundancy inherent in them. This process is called source coding. More specifically source coding is the process of converting the sequence of symbols generated by a source, in symbol sequences of the code (usually binary sequences), so as to remove the redundancy and the resulting compressed representation of messages.

Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying marginally important information and removing it [5].

Lossless compression examples of methods for source coding are Shannon coding [1], Shannon – Fano coding [3], Shannon – Fano – Elias coding [4], and Huffman coding [2], where the last one can been proven to be the optimum for a specific set of symbols with specific probabilities.

II. HUFFMAN CODING

A. Description - prerequisites

Huffman coding is the optimal code for a given probability of occurrence of the source symbols that can be obtained by means of a simple algorithm coding. It can be shown that no other algorithm can lead to the construction of a code with smaller average length codewords, for a given alphabet of the source. The outcome of the certain coding consists of a variable-length code table for encoding a source, where the variable-length code table has been derived in a particular way, based on the estimated probability of occurrence for each possible value of the source symbol [6].

B. The algorithm

According to the algorithm of Huffman, for the binary encoding of the source symbols follow the next steps:

1. The source symbols are arranged in decreasing transmission probability.
2. The last two symbols of the source with the lowest likelihood of producing combined into one, probability equal to the sum of the probabilities of the two symbols, resulting in the reduction of one of the plurality of conventional of all of the source alphabet.
3. Steps 1 and 2 are repeated until the source alphabet consists of only two symbols. In these two symbols assigned to 0 and 1 of the binary code.
4. A "0" and a "1" is assigned in place of one and the other symbol respectively, which in step 2 were merged into one. This step relates to all mergers.
5. The codewords of symbols formed by all the bits "0" and "1" associated with these symbols (from bottom to top), ie the digits are assigned directly to them or in merged symbols involved.

C. Example

The frequency of the letters in the English language (according to Wikipedia) is the following:
Assuming that we have to use the Huffman algorithm to encode the following subset of the English letters, having the above probabilities

\[ X = \{a, b, c, d, e\} \text{ and } P(X) = \{0.082, 0.015, 0.028, 0.043, 0.127\} \]

According to the algorithm, we first arrange the symbols in descending order of the transmission probability. The first column of the following table contains the symbols and the second column their chances. In the next step, the symbols with the least chance are combined with a probability equal to the sum of these probabilities. Then we re-arrange the symbols considering the joining of the last two.

In the next step, the symbols with the least chance are merged again and the remaining symbols are arranged again. We repeat the steps of merging and arranging of the symbols that are left until we finally end up with the merging of the last two symbols, having a probability equal to the sum of those of them. Starting from the last column of probability, we assign to them the symbols "0" and "1", respectively. In the prior probabilities column we also assign the symbols "0" and "1" next to the previous combined, etc.

Finally the resulting codes respectively for each letter of the chosen would be

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PROBABILITY</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>0.127</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>0.082</td>
<td>01</td>
</tr>
<tr>
<td>d</td>
<td>0.043</td>
<td>000</td>
</tr>
<tr>
<td>c</td>
<td>0.028</td>
<td>0010</td>
</tr>
<tr>
<td>b</td>
<td>0.015</td>
<td>0011</td>
</tr>
</tbody>
</table>

D. Limitations

Although Huffman’s algorithm is optimal for a symbol-by-symbol coding with a known input probability distribution, it is not optimal when the symbol-by-symbol restriction is dropped, or when the probability mass functions are unknown, not identically distributed, or not independent. Other methods such as arithmetic coding and LZW coding often have better compression capability: both of these methods can combine an arbitrary number of symbols for more efficient coding, and generally adapt to the actual input statistics, the latter of which is useful when input probabilities are not precisely known or vary significantly within the stream [6].

III. CASE STUDY

A. Description

In order to compare the results of the Huffman coding implementation between different types of human-produced texts, we have chosen to pick three representative documents of human activity. The first one is in the field of literature, a 1876 novel “The Adventures of Tom Sayer” by Mark Twain [7]. The novel is clearly indicative of the folklore surrounding life on the Mississippi River, around the time it was written. The second is a plain HTML document of the FrontPage from the BBC news channel website, on the 08-11-2012. It represents the modern spoken English in the western world. Finally the last document is the C language source code implementation of the Huffman algorithm itself. It represents a strictly technical document, with a limited set of words.

B. Implementation

Two programs were used in order to apply Huffman coding to the three documents. The first one, “letter_count.cpp”, accepts as input a file in text format and then counts the exact number of the letters and the spaces presented in the whole document. It also calculates the frequency of each symbol and writes these values to a text file named “freqs.txt”. The second, “huffman.c”, is an implementation of the Huffman algorithm, having as output the compressed binary code in a text file, calculating also the percentage of memory saved by means of the use of the algorithm.

C. Results

The first document, “The Adventures of Tom Sawyer”, was found to have a total of 379,055 letters and spaces altogether. The uncompressed text would have been encoded using 552 original bits in total for all letters and after the application of the Huffman coding the total bits used are 322. This yields to a 58.33% saved memory.

The second document, from www.bbc.com, was found to have a total of 88,687 letters and spaces altogether. The uncompressed text would have been encoded using 824 original bits in total for all letters and after the application of the Huffman coding the total bits used are 623. This yields to a 75.61% saved memory.

The third document, “huffman.c”, was found to have a total of 3,367 letters and spaces altogether. The uncompressed text would have been encoded using 144 original bits in total for all letters and after the application of the Huffman coding the total bits used are 78. This yields to a 54.17% saved memory.

IV. CONCLUSION

According to the results of the Huffman coding application to the three documents, we can say that the gain was greater to the bbc.com HTML document. This may be a result of the frequent appearance in this document of certain tags like <div>, <p>, etc. Also interesting is the fact that the amount of compression between the other two documents is almost equal. There has to be a further research with more documents to examine in order to end up with a safe conclusion.