A Study on Repeated Huffman Coding

Md. Bahlul Haider[1], Md. Jahid Hossain[2], Shafi Ullah Afrad[2],
S. M. Faisal Farman[2], Dr. M. Kaykobad[3]
[1] The University of Tokyo, [2] International Islamic University Chittagong (IIUC),
[3] Bangladesh University of Engineering and Technology (BUET)
bahlul_haider@mist.i.u-tokyo.ac.jp, raj_003144@yahoo.com, afradbd@yahoo.com,
smff_38@yahoo.com, kaykobad@cse.buet.ac.bd

Abstract

Since the discovery of the Huffman encoding scheme in 1952, Huffman codes have been widely used in efficient storing of data, image and video. Huffman coding has been subjected to numerous investigations in the past 60 years. Many techniques have been proposed since then. But still this is an important field as it significantly reduces storage requirement and communication cost. In this paper we have directed our works mainly to repeated Huffman coding which is a lossless coding technique. This paper studies performance of repeated Huffman coding. In repeated Huffman coding, Huffman encoding technique is applied on the binary file obtained by Huffman coding repeatedly. Since this technique is applied on a file with header, corresponding to the Huffman tree compression cannot be applied indefinitely. While it is expected that encoded message length will be smaller in every pass of repeated Huffman coding, nevertheless encoding the tree itself will be an overhead in each pass. So repetition count will depend upon how efficiently we can represent a Huffman tree. A memory efficient representation of a Huffman tree has been presented in this paper. Experimental results on ultimate compression ratios for different types of files have also been presented.

Keywords: Huffman Coding, Repeated Huffman Coding, Block Huffman Coding, Tree Clustering.

1. INTRODUCTION

The main objective of data compression techniques is to transform raw data into a form which is reduced in size but from which original data can easily be extracted. Mainly a compression technique focuses on reduction of storage requirements and communication cost over the network. Huffman coding counts occurrence of each symbol in the message, then constructs a Huffman tree based upon symbol probabilities and generates a code for each input symbol. Using these codes, Huffman tree and input symbols, the encoder builds a compressed output stream. Redundant information in a message takes extra bits to encode, and if we can get rid of that extra information, we will be able to reduce the size of the message. A Huffman algorithm requires two passes over the text. This causes delay when used for network communication. In file compression applications the extra disk access can slow down the operation. Faller [1] and Gallager [2] independently proposed a one pass scheme, which has been improved substantially by Knuth [3] for dynamic Huffman codes, usually known as FGK codes. J. S Vitter [4] has analyzed the dynamic Huffman codes and proposed an optimal algorithm. He also derived a tight upper bound for the dynamic Huffman codes. There are some practical problems in using traditional Huffman coding. One of the most prominent problems is that the whole stream must be read prior to coding. This is a major problem when file size is too large. Mannan and Kaykobad [5] solved the problem through introducing the block Huffman coding.

2. REPEATED HUFFMAN CODING

If Huffman coding technique can be applied effectively on a file again and again, then it is called repeated Huffman coding. While it is expected that encoded message length will be smaller in every pass of repeated Huffman coding, nevertheless encoding the tree itself will be an overhead in each pass. So repetition count will depend upon how efficiently we can represent a Huffman tree. If a Huffman tree can be represented efficiently in memory, repeated Huffman coding technique can be applied in an effective number of times. If we can do so, compression ratio will be increased.

Uncompressed data

Before compression

Tree header	Compressed data

After compression

Figure 01. Structure of compressed and uncompressed file.

At every pass of compression, repeated Huffman coding calculates the size of the compressed file. When it is found in any pass of compression that a compressed file size is greater than the previous compressed file then repeated Huffman coding stops its compression. Repeated Huffman coding is shown below with an example. Assume, a file contains total 100 times Y, 1 time Z and 102 times X. So the content of the file is YY...YZXX...X Required number of Bytes to represent the file is 100+1+102 = 203 Bytes. Now the Huffman tree is constructed based on the frequency of the characters. The Huffman tree is shown in figure 02.

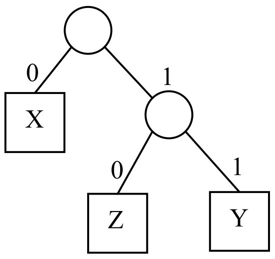

Figure 02. Huffman tree for first pass.

According to ordinary Huffman code, 0 is encountered for every left child traversing in the Huffman tree and 1 is encountered for every right child traversing. The Huffman code X is 0, Y is 11 and Z is 10.

Number of bytes to code the Huffman tree

= 4

Required bytes for compressed code

= Tree header + (100×2 + 2 + 102) / 8

= Distinct Symbols + Tree + 304 / 8

= 3 + 4 + 38

= 45 Bytes

So the required total number of bytes to represent the file after first pass of compression is 45.

Symbols	Tree Code	Code for 100 Y
YZX	0011000000110010	1111…….1

Code for 1 Z	Code for 102 X
10	000…..0

Figure 03. Compressed file format.

Some extra symbols will be generated to represent the compressed file after first pass of compression. Assume that the generated symbols are S1, S2, S3, S4, S5 and S6. For tree code S1, S2 and S3 can be defined by taking 4 bits for each symbol, because hexadecimal number is used to represent the tree code.

S1=0011
S2=0000
S3=0010

And S4, S5 and S6 symbols can be defined by taking 8 bits for each symbol where

S4= 11111111
S5 =10000000
S6 = 00000000

There are total 2 S1, 1 S2, 1 S3, 25 S4, 1 S5, 12 S6, 1 X, 1 Y and 1 Z in the source file of second pass. The frequency for each character is shown in Table 01.

Table 01. Frequency in second pass.

Character/Symbol	Frequency
S4	25
S6	12
S1	2
S2	1
S3	1
S5	1
X	1
Y	1
Z	1

Now the Huffman tree is constructed based on the above frequency table, shown below.

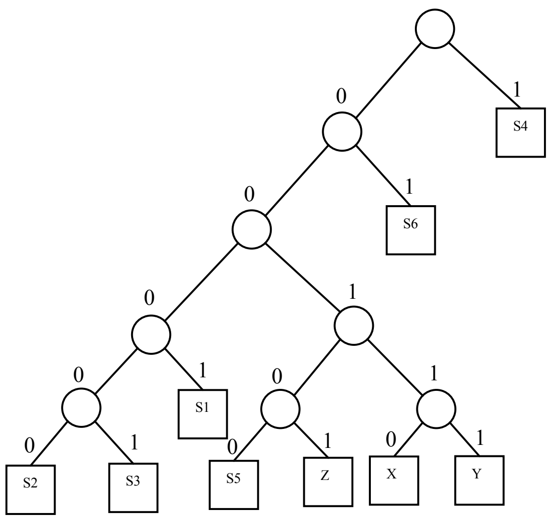

Figure 04. Huffman tree for second pass.

The Huffman code for corresponding character is given in Table 02.

Table 02. Code for each character in second pass.

Character/Symbol	Code
S4	1
S6	01
S1	0001
S2	00000
S3	00001
S5	00100
X	00110
Y	00111
Z	00101

No of Bytes to code the Huffman tree
= 16

Required Bytes for compressed code
= Tree Header + (25×1 +12×2 + 2×4 + 1×5+ 1×5+1×5+ 1×5+ 1×5+ 1×5 +1) / 8
= Distinct Symbols + Code for Tree + 88/ 8
= 9 + 16 + 11
= 36 Bytes

So the required total number of Bytes to represent the file after second pass of compression is 36. This pass cannot be continued because of degeneration. In third pass of compression the size of the compressed file is greater than the previous pass. So repeated Huffman coding stop compression.

2.1 Experimental Studies

The following experimental studies are obtained

Table 03. Experimental result.

Original File Size: 203 Bytes					
Different Approach	**Pass 1**	**Pass 2**	**Pass 3**	**Pure Huffman Compression Ratio (%)**	**Repeated Huffman Compression Ratio (%)**
Compressed file size	45	36	N/A	77.83	82.26
Compression Ratio	77.83	20	N/A		
Tree size	7	25	N/A		

Table 03 shows that the compression ratio for traditional and repeated Huffman coding is 77.83% and 82.26% respectively.

2.2 Algorithm: Repeated Huffman Coding

Step 1: Read each character from a file.
Step 2: Build the Huffman tree and code for each character.
Step 3: Compress this file using Huffman coding.
Step 4: Put the header and compressed data to output file.
Step 5: Calculate the size of the out put file.
Step 6: If
i) The size of the compressed file is smaller than the previous one then go to step 1.
ii) The size of the compressed file is greater than the previous one then exit.

2.3 Problem of Repeated Huffman Coding

An efficient representation of a Huffman tree is needed because the performance of the repeated Huffman coding depends on it. The number of repetition also depends on it.

2.4 Solution of these Problems

- Chung [6] gave a data structure requiring memory size only 2*n*-3 to represent the Huffman tree, where *n* is the number of distinct symbols.
- Chen et al. [7] improved the data structure further to reduce memory requirement to $\lceil 3n/2 \rceil + \lceil n/2 \lg n \rceil + 1$.
- Chowdhury et al. [8] improved memory requirements by considering circular leaf nodes (nodes with two adjacent external nodes). Since a Huffman tree has at most $\lceil n/2 \rceil$ circular leaf nodes, their memory requirement is $\lceil 3n/2 \rceil$ at worst case.

Traditional Huffman coding has many problems that are solved by dynamic Huffman coding, block Huffman coding and repeated Huffman coding. Arithmetic coding generates a single code for the whole source message. But this coding technique needs high computational power as well as arithmetic precision. Finally, repeated Huffman coding increase compression ratio.

3. HUFFMAN TREE REPRESENTATION

Here an existing technique of Huffman tree representation is used for repeated Huffman coding. Without keeping Huffman tree information it is not possible to decompress an encoded file. Coding of a Huffman tree is needed to get the tree from decoder side; otherwise extraction of data is not possible. Here coding of a Huffman tree means representation of the tree so that it can be reconstructed by decoder with minimal effort. There are many techniques to represent a Huffman tree. One of the technique is described below which is used for repeated Huffman coding.

3.1 Tree Representation Technique

In this method the sender sends a codeword of 0 and 1 for every distinct symbol. The sender also sends all information symbols with codeword. After receiving the tree header, the receiver reconstructs the Huffman tree from information symbols and codeword. Then it starts decoding using constructed Huffman tree.

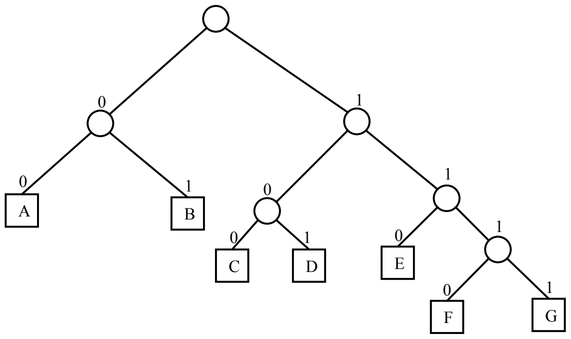

Figure 05. A Huffman tree.

The sender sends the following tree header with compressed data to the receiver.

Symbols	A	B	C	D	E	F	G
Code	00	01	100	101	110	1110	1111

Figure 06. Structure of tree header.

3.2 A Huffman Tree Extraction Process

When the receiver receives a tree header from the sender, it follows the following steps to reconstruct a Huffman tree. After reconstructing a Huffman tree the receiver will get the original message.

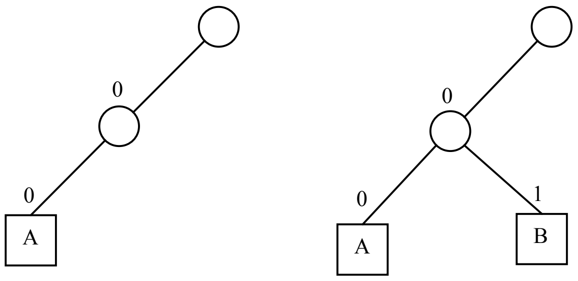

Figure 07. Reconstructing Huffman tree.

Receiver will reconstruct the following code.

Symbols	A	B	C	D	E	F	G
Code	00	01	100	101	110	1110	1111

Figure 08. Received tree header.

3.3 Space Complexity Analysis

Total number of distinct symbols = n

Space for all codes = n

Total space for Huffman tree representation

= Symbols + Code

= n + n

= 2n

A memory efficient representation of a Huffman tree reduces the overhead of every pass of repeated Huffman coding as well as a compression ratio is increased. Overhead reduction also increase the number of repetition count.

4. EXPERIMENTAL RESULT

This section shows the experimental data such as compression ratio, tree size, average code length, compressed file size and standard deviation of existing and proposed techniques. It also focuses on comparison between pure Huffman and repeated Huffman coding technique. The following tables show the compression ratio in every pass of compression.

Table 04. File description.

File #	File Name	Original Size (bytes)
1	TEST.TXT	3312164
2	RESULT.DOC	1063936
3	TC.EXE	1263536
4	MIG29.BMP	1104638
5	IIUC2300.PDF	625566
6	AVSEQ.DAT	30947068
7	BACH.DIR	5952908

Table 05. Compression ratio for repeated Huffman coding.

File #	Compression Ratio (%)					Repeated Huffman Compression Ratio (%)
	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5	
1	44.42	3.11	0.176	N/A	N/A	46.25
2	39.77	7.69	0.55	N/A	N/A	44.71
3	19.38	0.54	0.84	N/A	N/A	20.50
4	14.86	0.32	N/A	N/A	N/A	15.13
5	3.06	0.99	0.49	0.11	N/A	4.60
6	2.51	0.17	N/A	N/A	N/A	2.70
7	20.74	0.91	N/A	N/A	N/A	21.51

Table 06. Pure and repeated Huffman compression ratio (general files).

File Name	Original Size (Bytes)	Pure Huffman Coding	Repeated Huffman Coding
		Compression Ratio (%)	Compression Ratio (%)
TEST.TXT	3312164	44.42	46.25
RESULT.DOC	1063936	39.77	44.71
TC.EXE	1263536	19.38	20.50
MIG29.BMP	1104638	14.86	15.13
IIUC2300.PDF	625566	3.06	4.60
AVSEQ.DAT	30947068	2.51	2.70
BACH.DIR	5952908	20.74	21.51

Experimental results show that the proposed method provides a memory efficient compression. Repeated Huffman increase the compression ratio.

5. LIMITATIONS

In this paper only a concept of repeated Huffman coding and its implementation has been shown. An efficient Huffman tree or an optimal Huffman tree for repeated Huffman coding will enable to decrease size of a file dramatically. If the file size is very large, like image, audio, video files and if the frequency of all the ASCII characters from 0 to 255 is almost the same, then the file size will not be compressed as we predict. As we know that the Huffman tree is generated based on the frequencies of each character in a file and then the compressed file is written as ASCII character because each character of uncompressed file converted into bits using its corresponding codes generated by Huffman tree. In that there is gross chance for generation of code length for each character is more than 8 bits.

6. CONCLUSION

In this paper different types of Huffman coding have been studied. Also studied and implemented repeated Huffman coding. Comparison between the performance of Pure Huffman coding and repeated Huffman coding is also studied. This paper has mainly focused on how efficiency the Huffman coding technique can be improved through the repeated Huffman coding. It mainly focuses how repeated Huffman coding can increase the overall compression ratio. The compression technique discussed in here is lossless compression.

After all, it can be concluded that repeated Huffman coding increase the performance of pure Huffman coding. As well as the compression ratio also increases. This technique can be applied where a large volume of data needs to be stored. Suppose any large organization they need huge amount of storage to store their daily information, if they use this technique they can store their information in a minimum storage. Any mobile company needs Gigabytes of storage to store their daily call information. So they can use this technique, because it's a lossless coding technique there is no possibility of loosing their information. In this type of compression time complexity is not a mandatory issue.

7. RECOMMENDATIONS

Here lossless, variable length and repeated version of static Huffman coding is studied, implemented and tested for Huffman tree representation. Future works on Huffman tree representation and tree clustering algorithm can be carried on

- An optimal representation technique for a Huffman tree of *n* distinct symbols to get more compression ratio for repeated Huffman coding technique.
- More efficient decoding technique for the repeated Huffman coding can be devised. This decoding technique will extract the original information quickly. Less complex data structure for this technique is required.
- An effective clustering for the Huffman tree is also required. This clustering technique makes memory mapping faster, speed up search process and reduce memory wastage.

8. REFERENCES

[1] Faller, N., "An adaptive system for data compression", in record of the 7th Asilomer Conference on Circuits, Systems, and Computers. 1973, pp. 593-597.

[2] Gallager, R. G., "Variation on a theme by Huffman", IEEE Trans. Inf. Theory IT-24, 6(Nov. 1978), 668-674.

[3] Knuth, D. E., "Dynamic Huffman Coding", Journal of Algorithm 6, (1985) 163-185.

[4] Vitter, J. S., "Design and Analysis of Dynamic Huffman codes", J. ACM 34(4)(1987)825-845.

[5] Mannan, Mohammad Abdul and Kaykobad, M., "Block Huffman Coding", Computers and Mathematics with Applications.

[6] Chung, K.L., "Efficient Huffman decoding", Inform. Process. Lett. 61(1997)9799.

[7] Chen, H.-C., Wang, Y.-L. and Lan, Y.-F., "A memory-efficient and fast Huffman decoding algorithm", Inform. Process. Lett. 69(1999)119-122.

[8] Chowdhury, Rezaul Alam, Kaykobad, M. and King Irwin, "An efficient decoding technique for Huffman codes", Inform. Process. Lett. 81(2002)305-308.