2011 International Conference on Information and Network Technology
IPCSIT vol.4 (2011) © (2011) IACSIT Press, Singapore

Data Compression Scheme of Dynamic Huffman Code for Different Languages

Shivani Pathak[1], Shradha Singh[2], Smita Singh[3],Mamta Jain[4], Anand Sharma[5]

MITS, Lakshmangarh (*Rajasthan*)

[1]shivani.1326@gmail.com;[2]shradha18singh@gmail.com;[3]smi126.nov@gmail.com;[4]mamta11.jain@gmail.com;[5]anand_glee@yahoo.co.in

Abstract. Now a day's technology is growing in every aspect but the basic needs to save the memory and time utilization still remains. Whenever we deal with any kind of data (audio, video, text) we are bound to compress the data to minimize the space and time utilization, Data compression is a common requirement for most of the computerized applications, this paper examines lossless data compression on the Hindi, Hinglish and English texts and compares the performance based upon Huffman Coding.

Keywords: Data Compression; Dynamic Huffman code; Hindi and Hinglish (Hindi-English) Language.

1. Introduction

Compression is the art of representing the information in a compact form rather than its original or uncompressed form [1]. In other words, compression is to reducing the size of a file by removing redundancy in its structure. Compression is useful because it helps reduce the consumption of expensive resources, such as disk space or transmission bandwidth. Data Compression offers an attractive approach of reducing communication costs by using available bandwidth effectively.

Data Compression technique is divided into two broad categories:

- *Lossless Data Compression:* Lossless compression techniques reconstruct the original data from the compressed file without any loss of data, this means the information does not change during the compression and decompression of data. These kinds of compression algorithms are called reversible compressions since the original message is reconstructed by the decompression process [3]. It is used in many applications. For example, it is used in the ZIP file format and in the UNIX tool gzip.
- *Lossy Data Compression***:** Lossy compression techniques reconstruct the original message with loss of some information. It is not possible to reconstruct the original message using the decoding process, and is called irreversible compression [5].

The oldest and most widely used codes, ASCII and EBCDIC, are examples of block-block codes, mapping an alphabet of 64 (or 256) single characters onto 6-bit (or 8-bit) code words. The codes featured in this survey are of the block-variable, variable-variable and variable-block types. For example, in text file processing each character may constitute a message, or messages may be defined to consist of alphanumeric and non-alphanumeric strings[7]. The compression algorithms allow for records of different size and compressibility, along with File Marks, to be efficiently encoded into an output stream in which little or no additional control information is needed to later decode user data [8].

There are two methods to represent data before transmission.

Fixed length code: In this method it allows computers to efficiently process data which means that all characters are of equal length, even though they are not transmitted with equal frequency E.g. vowels, blanks used more often consonants, etc. ASCII is an example of a fixed length code. There are 100 printable characters in the ASCII character set and a few non printable characters, giving 128 total characters. Since

log 128 = 7, ASCII requires 7 bits to represent each character. The ASCII character set treats each character in the alphabet equally and makes no assumptions about the frequency with which each character occurs [9].

Variable length code: In this method most frequently transmitted characters are compressed; represented by fewer bits than least frequently transmitted and compressing data saves on data communications costs. A variable length code is based on the idea that for a given alphabet, some letters occur more frequently than others. This is the basis for much of information theory and this fact is exploited in compression algorithms. To use as few bits as possible to encode data without "losing" information. More sophisticated compression techniques can use compression techniques that actually discard information [9].

There are two dimensions along which each of the schemes discussed here may be measured, algorithm complexity and amount of compression. When data compression is used in a data transmission application, the goal is speed. 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, although the degree of compression is the primary concern [7]. Various lossless data compression algorithms have been proposed and used. Some of the main techniques in use are the Huffman Coding, Run Length Encoding, Arithmetic Encoding and Dictionary Based Encoding [6].This paper present the comparative study of the compression techniques applied on the different language text compression ratio.

1.1. Huffman Coding

Huffman Encoding Algorithms use the probability distribution of the alphabet of the source to develop the code words for symbols. The frequency distribution of all the characters of the source is calculated in order to calculate the probability distribution. According to the probabilities,the code words are assigned. Shorter code words for higher probabilities and longer code words for smaller probabilities are assigned. For this task a binary tree is created using the symbols as leaves according to their probabilities and paths of those are taken as the code words. Two families of Huffman Encoding have been proposed: Static Huffman Algorithms and Adaptive Huffman Algorithms. Static Huffman Algorithms calculate the frequencies first and then generate a common tree for both the compression and decompression processes [3].

The compression program maintains a count of how many times each symbol has occurred so far in the source text. To encode the next symbol, the symbol counts are used as estimates of the relative probabilities of the symbols and a table of Huffman codes based on these frequencies is constructed. The Huffman code in the table is used to encode the next symbol. (A minor detail that needs to be taken into account is that symbols in the alphabet that have not yet occurred in the source text must be assigned a non-zero probability estimate.) The decoding algorithm can re-create the same set of symbol frequencies from its de-compressed text and use the table to re-construct the same table of Huffman codes. Thus it can uniquely decode one symbol, update the frequency count of that symbol, update its table of Huffman codes and then decode the next symbol, and so on.

Avg L = L1 * P (1) + L2 * P (2) + ….. + Li * P (i)

Avg L = Σ Li * P (i)

Also the compression ratio as a measure of efficiency has been used. Comp. Ratio= Compressed file size /source file size *100 % The task of compression consists of two components, an encoding algorithm that takes a message and generates a "compressed" representation (hopefully with fewer bits) and a decoding algorithm that reconstructs the original message or some approximation of it from the compressed representation [5].

1.2. Demonstration Example

Example1: This example is derived from [9]

Suppose that we have a file of 100K characters. Each character is one of the alphabets from a to h. Since we have just 8 characters from a to h, we need just 3 bits to represent a character, so the space required to store the file is 300K. Now to reduce the space requires to store the file we can here use the concept of Variable Length Coding, and to do so more information about the file is required: the frequency with which

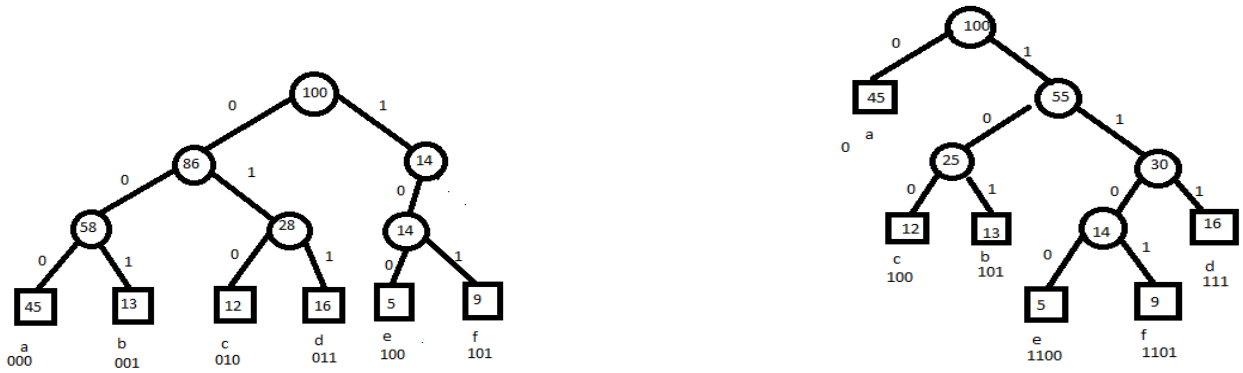
each character appears. The idea is to use fewer bits to store the most occurring characters and more bits to store the rare or least occurring characters.

Table 1

	a	b	c	d	e	f	g	H
Frequency	45K	13K	12K	16K	9K	5K	0K	0K
Fixed length code	000	001	010	011	100	101	110	111
Variable length code	0	101	100	111	1101	1100	-	-

For example, suppose that frequencies with which characters are occurring and following codes are given as shown in Table 1.Then the variable-length coded version will not take 300K bits but 45K* 1 + 13K* 3 +12K* 3 + 16K* 3 + 9K* 4 + 5K* 4 = 224K bits to store, a 25% saving. In fact this is the optimal way to encode the 6 character present.

Here we also include a concept of Prefix Codes which says that no code should be the prefix of any other code. The attraction of such codes is they help to encode and decode data easily. To encode, we need only concatenate the codes of consecutive characters in the message. So for example “cafe” is encoded as “100011001101”. To decode, we have to decide where each code begins and ends, since they do not have the same length. But this is easy, since, no codes share a prefix. This means we need only scan the input string from left to right and as soon as we recognize a code, we can print the corresponding character and start looking for the next code. In the above case, the only code that begins with “100...” or a prefix is “c”, so we can print “c” and start decoding “0110...” get “a”, etc.To see why the no common prefix property is essential, suppose that we tried to encode “e” with the shorter code “110” and tried to decode “1100”; we could not tell whether this represented “ea” or “f”.We can represent the decoding algorithm by a binary tree, where each left edge is assigned a 0 and each right edge is assigned a 1 and each leaf corresponds to the sequence of 0s and 1s traversed to reach it, i.e. a particular code. Since no prefix is shared, all legal codes are at the leaves and decoding a string means following edges, according to the sequence of 0s and 1s in the string, until a leaf is reached. Fig. 1 shows the tree for the above code.



Fixed Length Coding

Variable Length Coding

Fig. 1: Fixed Length v/s Variable length representation

2. Problem Statement

Upto what extent we can compress the text of different languages (Hindi, English, Hinglish) and to compare the compression ratios.

3. Our Approach

We took text of different languages: English, Hindi and their combination known as Hinglish and compared there compression ratio by using dynamic Huffman coding scheme of data compression

3.1. Comparision ratio of English language text

Suppose we want to decode the message "**ABRACADABRA**" using Huffman code. Table 2 shows the Huffman data compression and Fig. 2 shows the Huffman representation tree.

Table 2

Symbol	Occurrence	Probability	Codeword
A	5	5/11	0
B	2	2/11	110
R	2	2/11	10
C	1	1/11	1110
D	1	1/11	1111
Sum	11	1	

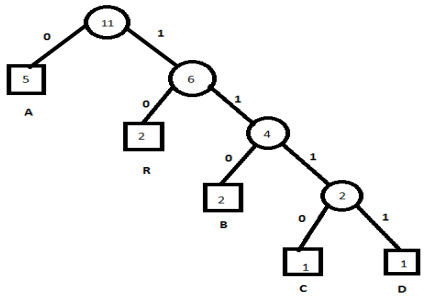


Fig.2

AVERAGE SEARCH LENGTH:

Avg L = Σ Li * P (i)

=5/11*1+2/11*3+2/11*2+1/11*4+1/11*4=23/11 = 2.090

Space required to store the original message=11*8=88 bits

Space required to store the decoded message=23 bits

Compression Ratio=26.136%

3.2. Comparision Ratio of Hinglish Language Text

Suppose we want to decode the message "**Kali ghata ka ghamand ghata**" using Huffman code. Table3 shows the Huffman data compression and Fig. 3 shows the Huffman representation tree.

Symbol	Occurrence	Probability	Code
K	2	2/27	1001
A	8	8/27	11
L	1	1/27	10000
I	1	1/27	10001
G	3	3/27	010
H	3	3/27	011
T	2	2/27	000
M	1	1/27	0010
N	1	1/27	00110
D	1	1/27	00111
Space	4	4/27	101
Sum	27	1	

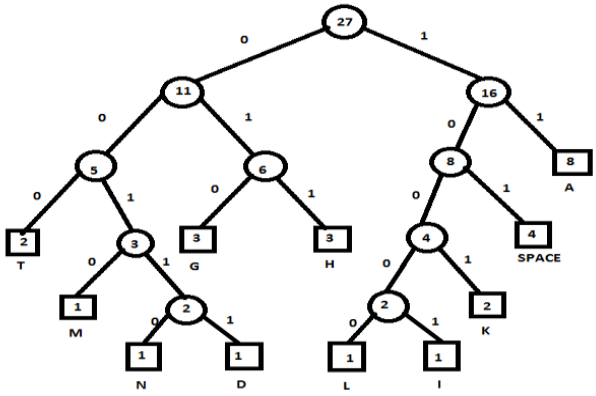


Fig.3

AVERAGE SEARCH LENGTH:

Avg L = Σ Li * P (i)

=2/27*4+8/27*2+(3/27+3/27+4/27+2/27)*3+(1/27+2/27)*4+(1/27+1/27+1/27+1/27)*5 = 84/27 = 3.11

Space required to store the original message=27*8 =216 bits

Space required to store the decoded message=84 bits

Compression Ratio=38.88%

3.3. Comparision Ratio of Hindi Language Text

Suppose we want to decode the message "**dkyh ?kVk dk ?keaM ?kVk**" using Huffman code. Table 4 shows the Huffman data compression and Fig. 4 shows the Huffman representation tree

Symbol	Occurrence	Probability	Code
d	1	1/19	11100
y	1	1/19	11101
?k	3	3/19	110
V	2	2/19	001
M	1	1/19	11110
e	1	1/19	11111
k	4	4/19	01
A	1	1/19	0000
h	1	1/19	0001
Space	4	4/19	10
Sum	19	1	

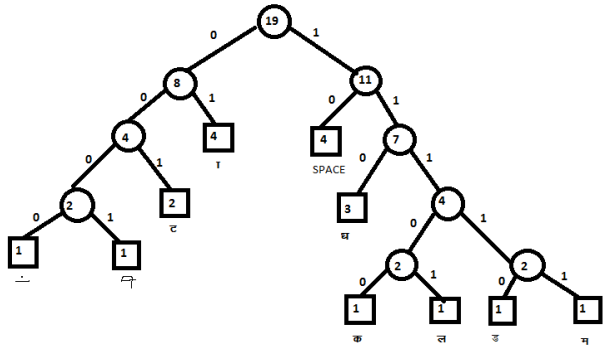


Fig.4

AVERAGE SEARCH LENGTH:

Avg L = Σ Li * P (i)

=(4/19+4/19)*2+(3/19+2/19)*3+(1/19+1/19)*4+(1/19+1/19+1/19+1/19)*5 =59/19 =3.105

Space required to store the original message=19*8=152 bits

Space required to store the decoded message=59 bits

Compression Ratio=38.81%

4. Conclusion

It has been found that the compression ratios of all the three texts are different and are gradually decreasing in the order Hinglish > Hindi > English and compressing the files saves space and reduces transmission time. Also we found that the compression ratios of Hinglish and Hindi text are almost equal and we can find out some more improvements to further reduce the compression ratios of these two in comparison to that of English and we can also try to find out the impact of file sizes on the compression ratios because we can observe here that if the file size is increased compression ratio can be improved as the occurrence of the characters will increase.

5. Acknowledgements

This work was supported by MITS Dean Mr.P.K.Das,We are thankful to him for his guidance.

6. References

[1] Pu, I.M., 2006, Fundamental Data Compression, Elsevier, Britain.

[2] Capocelli, M., R. Giancarlo and J. Taneja, 1986. Bounds on the redundancy of Huffman codes.IEEE Trans. Inf. Theory, 32: 854–857.

[3] Bentley, J.L., Sleator, D.D., Tarjan, R.E., and Wei, V.K. “A Locally Adaptive Data Compression Scheme.” CACM 29, 4 (April 1986), pp. 320-330.

[4] X. Kavousianos, E. Kalligeros, D.Nikolos, “Optimal Selective Huffman Coding for Test-Data Compression,” IEEE Trans.Computers, vol. 56, no. 8, pp. 1146-1152, Aug. 2007.

[5] Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.

[6] Kesheng, W., J. Otoo and S. Arie, 2006. Optimizing bitmap indices with efficient compression, ACM Trans. Database Systems, 31: 1-38.

[7] Vo Ngoc and M. Alistair, 2006. Improved word aligned binary compression for text indexing. IEEE Trans. Knowledge & Data Engineering, 18: 857-861.

[8] Cormack, V. and S. Horspool, 1987. Data compression using dynamic Markov modeling. Comput. J., 30: 541–550

[9] Kaufman, K. and T. Shmuel, 2005. Semi-lossless text compression. Intl. J. Foundations of Computer Sci., 16: 1167-1178.

[10] Dr. Muhammad Younus Javed and Mr. Abid Nadeem,"Data Compression Through Adaptive Huffman Coding Scheme" , IEEE-2000, Vol. II, pp.187-190,

[11] X. Kavousianos, E. Kalligeros, D. Nikolos, "Test Data Compression Based on Variable-to-Variable Huffman Encoding With Codeword Reusability," IEEE Trans.Comput.-Aided Des. Integr. Circuits Syst., vol. 27, no. 7, pp. 1333-1338, Jul. 2008.