International Journal of Computer Applications (0975 – 8887)
Volume 38– No.3, January 2012

A Comparative Study: Data Compression on TANGLISH Natural Language Text

S Sankar
Associate Professor, KCG College of Technology
Research Scholar, Hindustan University
Chennai, Tamilnadu, India

Dr. S Nagarajan
Professor and Head of Department of IT
Hindustan University
Chennai, Tamilnadu, India

ABSTRACT

In this age of information and in the era of distributed on-line and mobile computing, one thing is on the rise at an exponential rate is storage space for information. Growing office automation, digitizing libraries, on-line business transactions, and Meta data storage we need a huge storage space. Since more and more new users become a part of the Internet society the significance of data transmission develops to a great extent as never prior to. If data to be stored or transmitted represented efficiently this can be conquered. Data compression techniques are playing a vital role in representing the information. This paper investigates the use of lossless data compression on the Tanglish language text and compares the performance based upon Huffman coding.

Keywords

Lossless Data compression, Huffman coding, and Tanglish Language.

1. INTRODUCTION

Data compression is a method of bit reduction technique that uses a smaller amount of bits to represent information. The purpose is to reduce the amount of memory space, transmission bandwidth required. Also, it helps to increase data transmission rate over wired or wireless networks. On the other side, compressed data must be decompressed in order to make use of them. Compression schemes are generally classified into lossless and lossy. In lossless compression schemes what is compressed can be recovered without any loss of information. For that reason, they are appropriate to compress only textual data where meaning and clarity of the information is greatly anticipated. With lossy compression schemes, there will be some loss of information during decompression that is acceptable or unnoticed. So they are suitable for processing audio and video files where loss of resolution is ignored, depending on the preferred quality. The method characterized in this study is a kind of lossless compression used to compress a plain text.

A plain text is a form of highly unformatted text usually consists of alphanumeric and control characters. These characters are represented either in fixed length or variable length in the form of binary numbers 0 and 1 while storing and transmitting them. ASCII code and Unicode are fixed length coded character set tables that comprise numbers, letters, punctuation and various typographic and mathematical symbols and other characters. Each character in the set is represented by unique binary numbers. ASCII stands for American Standard Code for Information Interchange contains a set of 128 characters where the capital letter ‘A’ has decimal number 65 and is not stored as it is rather as 1000001. A character in the ASCII table has 7 bit length. Unicode is a Universal character set table contains 65535 characters that cover almost all the characters, punctuations, and symbols in the world. UTF-8 (Unicode Transformation Format in 8 bits) is a type of Unicode character set where each character has 1 to 4 bytes long, for example, a Tamil language character ‘அ’ (read as ‘a’) has decimal value 2949 and is stored as 11100000 10101110 10000101. In a fixed length code, each character has the same length, so it is possible to calculate exactly where each character begins therefore it is quicker to find a particular character for the decompression but it occupies more memory.

Consider a message M having symbol set {abcaabcaaabbc}. If it is represented in fixed length code as a=00, b=01 and c=10 then each bit string is a codeword for a symbol. Now the message can be encoded as 00011000000110000000010110 where the total number of bits is 26. If a message contains n number of distinct symbols then each symbol will require exactly [$\log_2$ n] length of bits. In the example just given, number of distinct symbols in the message is n=3, as a result we need [$\log_2$ 3] = 2 bits to represent each symbol. Grouping of n bit length from the beginning of the encoded binary string will result a symbol. By scanning from the beginning the first two bits 00 results a symbol ‘a’ and the next two bits 01 result a symbol ‘b’ and so on. If n bits are required to encode a symbol, then 2n distinct symbols can be encoded. In ASCII, each character has exactly 7 bit length, so that there are 27=128 distinct symbols. Total number of bits required to encode complete message can be computed as, length of the message * number of bits per symbol. Reducing the length of the code is very important since the amount of time required for the data transmission always proportional to the number of bits required to encode it.

A variable length code comes with the solution for this where each character will be represented with different length of code. Suppose the symbols represented as a=0, b=10, and c=01 then the entire message is encoded as 01001001001000 101001 where the total number of bits are 20 which is 23.08% less compared to fixed length variable code. But the problem with this is how to recognize the end of one codeword and the beginning the next one during decode. The first bit is 0 and the next bit is 1 so whether to decode it as 0 which is ‘a’ or as 01 which is ‘c’. Thus this is ambiguous. To prevail it a prefix code is followed. A prefix code is a variable length code it uses prefix rule where no codeword is a prefix of another. Once a certain bit pattern is assigned as codeword of a symbol, no other codeword should start with that bit pattern. If a bit pattern 0 was assigned as the codeword of ‘a’, then no other codes could start with 0. Therefore the codeword for the symbol ‘c’ should not be 01 rather than it can be 11. If so, all codeword can be unambiguously decodable since once we get a match, there is no longer codeword that can also match. Various lossless variable length code algorithms have been proposed and used. Some of the techniques in use are the Huffman Coding, Run Length Encoding, Arithmetic Encoding and Dictionary Based Encoding [8]. This paper studies the efficiency of codeword created for the Tanglish language text using Huffman coding.

2. HUFFMAN CODING

The Huffman code algorithm generates a prefix and variable length codeword for a symbol based on the symbol probability distribution p_i, where i = 1, 2, 3. . . n. The frequency distribution of all the symbols of the source is calculated in order to calculate the probability distribution. According to the probabilities, the codeword for each symbol are assigned. It assigns shorter codeword for higher probability symbols and longer codeword for smaller probability symbols [8]. This algorithm is optimal in the sense that the average number of bits required to represent a symbol is minimized, subject to the constraint that the codeword satisfies the prefix rule, as defined above. Average number of bits required to encode the message can be computed as, $L = \sum_{i=1}^{n} l_i * p_i$, where l_i is the codeword length of a symbol and p_i is the probability of a symbol [13]. The compression ratio as a measure of efficiency has been considered and can be calculated as, Compression ratio= (Compressed file size/Source file size) * 100 % [8].

The idea behind the algorithm is, first construct an optimal binary tree so-called Huffman tree which adopts a greedy approach. The greedy method suggests construct a solution through a sequence of steps, considering one input at a time. At each step, make a locally optimal choice among the currently available all feasible choices; once made it cannot be changed on subsequent steps and that choice may lead to the development of the globally optimal solution. An optimal merging pattern is followed to construct the Huffman tree. In which sort the symbols in increasing order based on their probabilities ($p_1 \le p_2 \le p_3 \le . . . \le p_n$). At each step merge two smallest probability symbols together. If any two symbols have equal probabilities, interchange them based on their appearance in the ASCII or Unicode table. When more than two sorted symbols are to be merged together, the merge can be accomplished by repeatedly merging sorted symbols in pairs. The leaf nodes represent the given symbols and are called as external nodes. The remaining nodes are called as internal nodes. Each internal node has exactly two children and its value is obtained by merging the probabilities of its two children. Tree is built in a bottom up fashion [14].

2.1 Algorithm for Huffman Tree

Input: Symbol set with their respective probability distribution. (Probability of a symbol=frequency of a symbol / total number of symbols in the message)

Output: Huffman Tree T^+

```
1. Sort the symbol set based on their probability in non
   decreasing order
2. Construct a forest tree F for the given symbol set where
   each tree having only one node include the symbol and its
   probability
3. Repeat {
   3.1 Choose the nodes with the minimum and next to
       minimum probabilities respectively
   3.2 Create a new node $T^+$. Node value for $T^+$ = sum
       (minimum probability node, next to minimum
       probability node)
   3.3 Attach a minimum probability node on left to $T^+$ and
       next to the minimum probability node on right to $T^+$
   3.4 Assign 0 to left branch and 1 to the right branch.
       Insert $T^+$ into F
   3.5 Sort the tree F in non decreasing order
4. } Until (no more than one node tree in F)
5. Now F has only one tree $T^+$. Output $T^+$.
```

2.2 Encoding

A path from root of T^+ to the corresponding leaf node defines codeword for a particular symbol. Right margins should be justified, not ragged.

2.3 Algorithm for Decoding

Input: Codeword generated during encode
Output: Symbol

```
1. Start from the root of the tree $T^+$.
2. Examine the first bit in the input
3. If it is 1, move to the left child.
4. If it is 0, move to the right child.
5. If it is leaf node then output its symbol.
6. If it is not a leaf node then
   6.1 Examine the next bit in the input
   6.2 Go to step 3 and proceed
```

3. TANGLISH

Tanglish is a dialect in which a sentence is formed in Tamil borrows words from English. It is an informal language. If it is difficult to find appropriate Tamil words in writing or verbal communication, identical word in English will be used in place. Now it is becoming a fashionable language used by Tamil speaking people in Tamilnadu, a southern state of India. It is a new hybrid language where a sentence is formed by mixing of linguistic features of both Tamil and English even though both has a different syntactical pattern, has found its way into the media - electronic and print [7].

A sentence in Tanglish language can be constructed in three different forms:
Form 1: Inserting English word in the place of the Tamil word in a sentence usually found in Tamil magazine and oral communication
Form 2: Representing English word in Tamil usually found in web pages
Form 3: Representing Tamil word in English frequently used in SMS (Simple Message Service) messages and web pages

4. PROPOSED APPROACH

This paper applies the Huffman code technique on the above said forms and compares the compression ratio to determine which form is an efficient. A sentence, “car is breakdown” is taken as a sample message for the demonstration. This sentence can be written in Tanglish as:
Form 1: car breakdown aagivitathu
Form 2: kaar piraegtovun aagivitathu
Form 3: vakanam paLuthagi vitathu
The following section illustrates the creation of Huffman tree and the codeword for these different forms of representation.

4.1 Data compression on Form1

Figure 1 shows Huffman tree generated for the message “car breakdown aagivitathu”. Table 1 shows the codeword generated for the same.

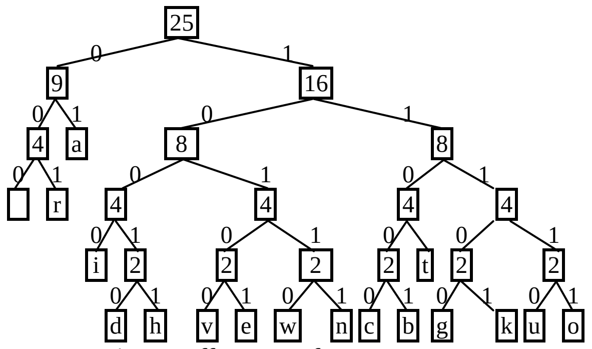

Fig 1: Huffman tree for Form1 message

The encoded message is now,
11000010010001100100110101011110110010111111011010
1110000010111100100010100100011010111011001111110.
Average number of bits required to encode = 5 / 25 * 2 + 2 / 25 (3 + 3 + 4 + 4) + 1 / 25 (5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5)
= 98 / 25 = 3.92
Space required for actual message= 25 x 8 = 200 bits
Space required for encoded message= 98 bits
Compression ratio= 49%

Table 1. Codeword for Form1 message

Symbol	Frequency	Probability	Codeword
a	5	5/25	01
	2	2/25	000
r	2	2/25	001
i	2	2/25	1000
t	2	2/25	1101
d	1	1/25	10010
h	1	1/25	10011
v	1	1/25	10100
e	1	1/25	10101
w	1	1/25	10110
n	1	1/25	10111
c	1	1/25	11000
b	1	1/25	11001
g	1	1/25	11100
k	1	1/25	11101
u	1	1/25	11110
o	1	1/25	11111

4.2 Data compression on Form2

Figure 2 illustrates the Huffman tree generated for encoding the message “kaar piraegtovun aagivitathu”. Table2 information is used to encode this message. The encoded message is,
10100000011101111101010111110001011110000101011011
00100111010111100001000011110001101000010110111001
Average number of bits required to encode = 6 / 28 * 2 + 3 / 28 (3 + 3) + 2 / 28 (4 + 4 + 4 + 4 + 4) + 1 / 28 (5 + 5 + 5 + 5 + 5 + 5)
= 100 / 28
= 3.57
Space required for actual message= 28 x 8 = 224 bits
Space required for encoded message= 100 bits
Compression ratio= 44.64%

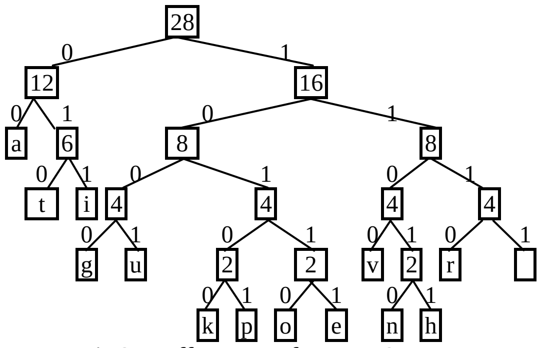

Fig 2: Huffman tree for Form2 message

Table2 lists the codeword generated for the above message.

Table 2. Codeword for Form2 message

Symbol	Frequency	Probability	Codeword
a	6	6/28	00
t	3	3/28	010
i	3	3/28	011
g	2	2/28	1000
u	2	2/28	1001
v	2	2/28	1100
r	2	2/28	1110
	2	2/28	1111
k	1	1/28	10100
p	1	1/28	10101
o	1	1/28	10110
e	1	1/28	10111
n	1	1/28	11010
h	1	1/28	11011

4.3 Data compression on Form3

Figure 3 shows the Huffman tree constructed for the message “vakanam paLuthagi vitathu”. Table 3 shows the codeword generated for the same. The actual message is now encoded as,
11110110100011010101111010011011001111001100100000
0110111110100111111101100011000001100.
Average number of bits required to encode = 6 / 25 * 2 + 3 / 25 * 3 + 2 / 25 (3 + 3 + 4 + 4 + 4) + 1 / 25 (5 + 5 + 5 + 5 + 5 + 5)
= 87 / 25
= 3.48

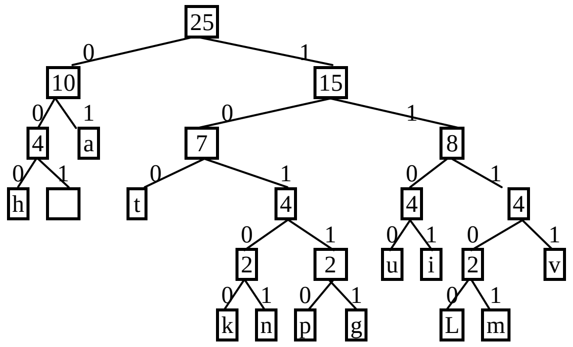

Fig 3: Huffman tree for Form3 message

Space required for the actual message
= Total number of symbols * 8 bits per symbol
= 25 x 8
= 200 bits
Space required for encoded message= 87 bits
Compression ratio= 43.5%

Table 3. Codeword for Form3 message

Symbol	Frequency	Probability	Codeword
a	6	6/25	01
t	3	3/25	100
h	2	2/25	000
	2	2/25	001
u	2	2/25	1100
i	2	2/25	1101
v	2	2/25	1111
k	1	1/25	10100
n	1	1/25	10101
p	1	1/25	10110
g	1	1/25	10111
L	1	1/25	11100
m	1	1/25	11101

5. RESULTS AND DISCUSSION

Ten different plaint text files written in English with different sizes are considered. The file sizes are 2524 bytes, 2023 bytes, 3334 bytes, 2867 bytes, 4109 bytes, 2878 bytes, 4745 bytes, 5189 bytes, 5432 bytes, and 4901 bytes respectively. The content of each file is translated into above mentioned three different forms in Tanglish language and Huffman coding is applied on each of them and the results are compared.

5.1 Results

Table 4 shows Huffman coding results where it summarizes the results of average number of bits required to encode and compression ratio for each file. According to the results shown in Table4, for file 1 and 10 the algorithm generates higher compression ratio in Form1. This happens due to the direct use of actual English words in the place of Tamil words at high rate. The files 4 and 6 has lower compression ratio due to lesser use of actual English words.

Table 4. Huffman coding results

File	Average number of bits required to encode			Compression ratio (%)		
	Form 1	Form 2	Form 3	Form 1	Form 2	Form 3
F1	3.92	3.57	3.48	49.00	44.64	43.50
F2	3.50	3.53	3.57	43.75	44.08	44.64
F3	3.55	3.53	3.47	44.32	44.12	43.38
F4	3.32	3.45	3.22	41.52	43.36	39.86
F5	3.95	3.91	3.77	49.39	48.94	47.14
F6	3.21	3.35	3.50	40.18	42.01	43.75
F7	3.87	3.84	3.81	48.40	48.06	47.70
F8	3.84	3.72	3.88	48.04	46.59	48.56
F9	3.90	3.87	3.87	48.84	48.38	48.38
F10	4.00	3.98	3.67	50.00	49.75	45.91

For all other files the compression ratio is almost same range in Form1. In Form2, the files 5, 7, 9 and 10 has higher compression ratio where it is found a large amount of actual English words are written in Tamil words. In Form3, most of the file has a lower compression ratio since average number of bits required to encode message is less for each file than for the same in Form 1 and 2.

5.2 Discussion

From the figure 4, it has been found that compression ratio gradually decreasing in the order of Form1>Form2>Form3 and compressing the files saves the disk space and transmission time. Also, we found from Table 1, 2 and 3 that Huffman code generates a short codeword for the symbol which has higher probability and lengthy codeword for the symbol which has lower probability.

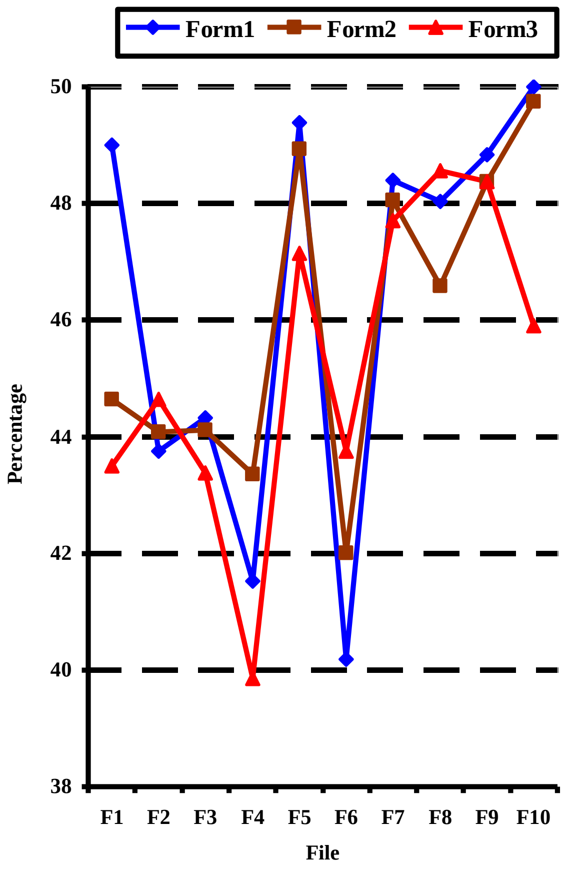

Fig. 4 Compression Ratio

6. CONCLUSION

A lossless data compression algorithm is carried out on different file. Each file is translated in three different forms of Tanglish language and Huffman coding is applied on each of them. The resulting compression ratios are compared. We can observe that often placing directly actual English words in the place of Tamil words in a Tanglish Language sentence turn out a better result than either representing the English words in Tamil or Tamil words in English in terms of data compression. Also, it is observed that Huffman code produces average number of bits required to encode a message is higher for Form1 than others.

7. REFERENCES

[1] Abu Shamim Mohammad Arif, Asif Mahamud, and Rashedul Islam. 2009. An enhanced static data compression scheme of Bengali short message. International Journal of Computer Science and Information Security. Volume 4. No. 1 & 2.

[2] Asher, R.E., and Annamalai, E. 2002. Colloquial Tamil- The Complete Course for Beginners. Routledge Publication. ISBN 0-203-99424-8.

[3] Brent, R. P. 1987. A linear algorithm for data compression. The Australian Computer Journal. Volume 19. No.2.

[4] David R. Mcintyre, and Michael A. Pechura. 1985. Data compression using static Huffman code-decode tables. Communications of the ACM. Volume 28. Issue 6.

[5] Haroon Altarawneh, and Mohammad Altarawneh. 2011. Data Compression Techniques on Text Files: A Comparison Study. International Journal of Computer Applications. Volume 26. No. 5.

[6] Huffman, D. A. 1951. A method for the construction of minimum redundancy codes. Proceedings IRE. Volume 40. Pages 1098-1101.

[7] Kanthimathi, K. 2009. Tamil-English Mixed Language Used in Tamilnadu. The International Journal of Language Society and Culture. Issue 27. ISSN 1327-774X.

[8] Kodituwakku, S.R., and Amarasinghe, U.S. Comparison of lossless data compression algorithms for text data. Indian Journal of Computer Science and Engineering. Volume 1. No. 4. Pages 416-425.

[9] Mamta Sharma. 2010. Compression Using Huffman Coding. International Journal of Computer Science and Network Security. Volume 10. No.5.

[10] Mohammed Rafiul Hassan, and Baikunth Nath, 2005. Data compression using Huffman coding – a novel approach. International Conference on Applied Computing. ISBN: 972-99353-6-X.

[11] Popuri Ramesh Babu, Gonuguntla Rama Swamy, Daruvuri Ravi Kiran, and Devireddy Srinivasa Kumar4. 2009. A novel approach for data compression in E- mail. International Journal of Research and Reviews in Applied Sciences. Volume 1. Issue 1. ISSN: 2076-734X, EISSN: 2076-7366

[12] Sayood. K. 2000. Introduction to Data Compression. Second Edition. Morgan Kaufmann publications.

[13] Viswanath. K. 2002. General Article: Communication Information. Resonance. Volume 7. No. 2. Pages 26-32. DOI: 10.1007/BF02867266.

[14] William Ford, and William Topp. 2002. Data Structure with C++ using STL. Prentice Hall. ISBN: 0-13-085850-1.

[15] Ziv, J. and Lempel, A. 1977. A universal algorithm for data compression. IEEE Transactions on Information Theory. IT-23(3). Pages 337-343.