A New Compression Method Strictly for English Textual Data

Sabina Priyadarshini
Department of Computer Science and Engineering
Birla Institute of Technology

Abstract - Data compression is a requirement these days as it makes the communication of data faster. There are many techniques of data compression. The paper proposes a lossless method of data compression which is very simple and is very fast and efficient. It is a fixed length encoding of data and is very fast to decompress in comparison with the existing lossless compression techniques.

Keywords: compression, English data, lossless compression

I. INTRODUCTION

All english characters take 8 bits of memory to be stored. The letter “A” has an ascii value 41. The letter “B” has ascii value 42 and so on. The letters and their corresponding 8 bit ascii values are given in the following table.

Table 1

Character	Ascii Hexadecimal	8 Bits representation
@	40	0100 0000
A	41	0100 0001
B	42	0100 0010
C	43	0100 0011
D	44	0100 0100
E	45	0100 0101
F	46	0100 0110
G	47	0100 0111
H	48	0100 1000
I	49	0100 1001
J	4A	0100 1010
K	4B	0100 1011
L	4C	0100 1100
M	4D	0100 1101
N	4E	0100 1110
O	4F	0100 1111
P	50	0101 0000
Q	51	0101 0001
R	52	0101 0010
S	53	0101 0011
T	54	0101 0100
U	55	0101 0101
V	56	0101 0110
W	57	0101 0111
X	58	0101 1000
Y	59	0101 1001
Z	5A	0101 1010
[	5B	0101 1011
\	5C	0101 1100
]	5D	0101 1101
^	5E	0101 1110
-	5F	0101 1111

Compressing the data helps in faster communication of information. If the number of bits required to encode the data can be reduced, more information can be sent in fewer bits. There are two types of compression- one being lossy and the other being lossless. Lossless compression reduces bits by identifying and eliminating statistical

redundancy [1].No information is lost in lossless compression. Lossy compression reduces bits by identifying unnecessary information and removing it. The design of data compression schemes involve trade offs among various factors, including the degree of compression, the amount of distortion introduced (eg, when using lossy data compression) and the computational resources required to compress and uncompress the data.

A simple method to compress English textual data has been proposed in this paper which gives a compression ratio of 5/8 which is 62.5% and a space savings of 1- (5/8) which is 37.5%. The compression is lossless in nature and does not even depend on the probability of occurrences of the letters within the textual data. This method has been discussed in section III. An example illustration is given in section IV and a comparison with Huffman coding, run length encoding and arithmetic coding has been done in section V. Section VI gives the decompression technique and section VII concludes the paper.

II. LITERATURE SURVEY

Huffman coding is a lossless compression technique of data compression. The term refers to the use of variable length code table for encoding a source symbol where the variable length code table has been derived in a particular waybased on the estimated probability of occurrence for each possible value of the source symbol [2]. It was developed by David A. Huffman while he was a Ph.D student at MIT and published in the 1952 paper" A Method for the Construction of Minimum Redundancy Codes" [3]. Huffman coding is a form of statistical coding. All characters do not occur with the same frequency. But all characters occupy the same amount of space which is 1 byte. Huffman came up with the idea that code word lengths will no longer be fixed like ASCII. Code word lengths will vary and will be shorter for more frequently used characters. Huffman coding algorithm has the following steps [4]:-

1. Scan text to be compressed and tally occurrence of all characters.
2. Sort or prioritize characters based on number of occurrences in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.

For decoding the file, the Huffman tree must be sent along with the encoded file. The tree is prepended to the file. For smaller files, passing this tree takes a lot of space and is a big hit on compression. Also, Huffman coding depends on frequency of occurrence of letters in the text. If some characters are repeated more often in the text, then they can be sent in fewer number of bits. But if the text does not contain repeating characters, then, the advantage from Huffman coding is less.

Arithmetic coding is a form of entropy encoding used in lossless data compression. When a string like " hello there" is converted to arithmetic coding, frequently used characters will be stored with fewer bits and not-so frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. Arithmetic coding differs from other forms of entropy encoding like Huffman coding in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message to a single number, a fraction n where (0.0<=n<1.0). Arithmetic coding has a higher computational complexity then Huffman coding [5].

Run length encoding is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than the original run. This is most useful on data that contains many runs, for example, simple graphic images such as icons, line drawings and animations. It is not useful with files that don't have many runs as it could greatly increase the file size. Suppose the text is wwwwwwwwwtttt, then, we can write it in encoded form as w9t4 [6]. But if the text is swtt, then it will be written as s1w1t2.

III. THE PROPOSED METHOD

The proposed method replaces each 8 bit value (1 byte character) by a 5-bit value. This method can be used only on data that is textual. It will not be able to encode numeric data. The Table II shows how each character would be represented by a 5-bit value rather than an 8-bit ascii value. All the uppercase letters lie in the range 41-5A. 5B through 5F and 40 are used to represent special characters. The proposed method makes use of 40 and 5B through 5F to represent '-', comma, full stop, space, double quotes and single quotes. That means, in our method, we represent a comma by a '[', a full stop by a '\', a space by a ']' sign, double quotes by a '^' and a single quote by a '-'. The sign '@' is used to represent '-'. A' 4' in ascii which evaluates to 0100 can be represented by a '0' and a 5 in ascii hex can be represented by a '1'. In this way, if 'A' has ascii value 41, it is represented by 1 byte as 0100 0001. We represent it by encoding the first 4 bits (which is a 4) with a 0. We replace the first four bits by a zero and 'A' can be represented as 00001. In the same way, 'B' is originally represented as 42 which is 0100 0010. We can again

represent the first 4 bits (which represents a 4) with a zero and write ‘B’ as 00010. In the same way,’P’ is originally represented as 50 which is 01010000. We can represent first 4 bits (which is a 5)as ‘1’ and write P as 10000. In this way, each character can be represented by 5 bits and not 8 bits, thereby bringing about compression. Since, all the English capital letters lie in the range 40 to %F, so every 4 can be replaced by a ‘0’ and every 5 can be represented by a ‘1’. Bit fields in C language can be used for compression.

The steps of the proposed method are as follows:

i) Convert all text into uppercase.

ii) Replace a comma with a ‘[‘, a full stop by a ‘\’, a space by a ‘]’, a double quote by a ‘^’ , a single quote by a ‘-‘ and a ‘-‘ with a ‘@’.

iii) Substitute each character with its 5 bit equivalent code as given in Table II.

Table II.

Character	Ascii value	Used to rpresent	5 bits representation
@	40	‘-‘ (hyphen)	0 0000
A	41	‘A’ or ‘a’	0 0001
B	42	‘B’ or ‘b’	0 0010
C	43	‘C’ or ‘c’	0 0011
D	44	‘D’ or ‘d’	0 0100
E	45	‘E’ or ‘e’	0 0101
F	46	‘F’ or ‘f’	0 0110
G	47	‘G’ or ‘g’	0 0111
H	48	‘H’ or ‘h’	0 1000
I	49	‘I’ or ‘I’	0 1001
J	4A	‘J’ or ‘j’	0 1010
K	4B	‘K’ or ‘k’	0 1011
L	4C	‘L’ or ‘l’	0 1100
M	4D	‘M’ or ‘m’	0 1101
N	4E	‘N’ or ‘n’	0 1110
O	4F	‘O’ or ‘o’	0 1111
P	50	‘P’ or ‘p’	1 0000
Q	51	‘Q’ or ‘q’	1 0001
R	52	‘R’ or ‘r’	1 0010
S	53	‘S’ or ‘s’	1 0011
T	54	‘T’ or ‘t’	1 0100
U	55	‘U’ or ‘u’	1 0101
V	56	‘V’ or ‘v’	1 0110

W	57	'W' or 'w'	1 0111
X	58	'X' or 'x'	1 1000
Y	59	'Y' or 'y'	1 1001
Z	5A	'Z' or 'z'	1 1010
[	5B	',' (comma)	1 1011
\	5C	'.' (full stop)	1 1100
]	5D	' '(space)	1 1101
^	5E	'"' (double quotes)	1 1110
-	5F	''(single quote)	1 1111

IV. ILLUSTRATION

We can compress the following sentence:-
"This is an example of compression. "
First convert it into uppercase :
THIS IS AN EXAMPLE OF COMPRESSION.
Then, write ascii value of each character as given in table II.

T	54	10100
H	48	01000
I	49	01001
S	53	10011
Space	5D	11101
I	49	01001
S	53	10011
Space	5D	11101
A	41	00001
N	4E	01110
Space	5D	11101
E	45	00101
X	58	11000
A	41	00001
M	4D	01101
P	50	10000
L	4C	01100
E	45	00101
Space	5D	11101
O	4F	01111
F	46	00110
Space	5D	11101
C	43	00011
O	4F	01111
M	4D	01101
P	50	10000
R	52	10010
E	45	00101
S	53	10011
S	53	10011
I	49	01001

O	4F	01111
N	4E	01110
.	5C	11100

This text would require 5X34=170 bits to encode. Originally, this text would require 272 bits. So, we have saved 272-170=102 bits. The compression ratio is 62.5%. The space saving is 37.5%.
If we encode this same text using Huffman coding, it would require 137 bits but with Huffman coding, we have to transfer the entire Huffman tree along with the encoded data for lossless decompression, which would require more than 33 bits. So, the space saving would not be good.

V. ADVANTAGES OF THE TECHNIQUE

1. The method can compress text of any kind. It does not have to care about frequency of occurrence of characters in the text. The compression techniques like Huffman coding, arithmetic coding and run length encoding make use of frequency of occurrence of characters within the text for compression.
2. In Huffman coding, for decoding the file, the entire Huffman tree must be sent along with the encoded file. The tree is prepended to the file. For smaller files, passing this tree takes a lot of space and is a big hit on compression [4]. Also, Huffman coding depends on frequency of occurrence of letters in the text. If some characters are repeated more often in the text, then they can be sent in a fewer number of bits. But if the text does not contain repeating characters, then, the advantage from Huffman coding is less.
3. Both Huffman coding and arithmetic coding have higher computational complexity than the proposed method. The proposed method is faster and simpler as it is based on simple substitutions.
4. The proposed method uses fixed length encoding and not variable length encoding. So, decoding is faster and easier.

The entropy is the average number of bits per character of the message required to encode the message so that it can be fully recovered [7]. The entropy of the proposed method is 5 bits per character.

VI. DECOMPRESSION

The following are the steps for decompression:

(i) Divide the received bits-sequence to chunks of 5 bits each.
(ii) Replace each 5 bit data into its equivalent 8 bit value using the tables I and II.
(iii) Replace each 8-bit value so obtained into its corresponding character form using the tables.
(iv) Replace a '[' with a comma, a '\' with a full stop, a ']' by a space, a '^' by a double quote, a '-' by a single quote and an '@' by a '-'.
(v) We get the uncompressed text.

VII. CONCLUSION

A method of data compression has been proposed that is both fast and lossless. It gives a compression ratio of 62.5% and space saving of 37.5%. The method is better than Huffman coding and arithmetic coding in terms of computational complexity. It also performs better than run length encoding as the compression is not dependent on the frequency of occurrence of characters within the text. The only limitation is that it compresses pure textual data that includes English text characters, comma, full stop, space, double quote, single quote and a hyphen.

REFERENCES

[1] en.wikipedia.org/wiki/Data_ Compression
[2] en.wikipedia.org/Huffman_coding
[3] David A. Huffman, "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the IRE, September 1952, pp. 1098-1101.
[4] www.cs.nyu.edu/~melamad/courses/102/lectures/huffman.ppt.
[5] en.wikipedia.org/wiki/ Arithmetic_coding.
[6] en.wikipedia.org/wiki/Run length encoding
[7] ta.ramk.fi/~jouko.teeriaho/shanon.pdf