A Modified Approach to Huffman Algorithm for Unequal Bit Costs

Abstract—**Classical Huffman coding is an encoding scheme that creates variable length code to compress data. It has very good performance over data compression for many current systems. The goal of the Huffman algorithm is to encode a set of characters given their frequency over a given alphabet. It is necessary that the encoding is prefix free to keep the code decodable. But better encoding is possible if we consider different cost for different symbols of the alphabet. In the case of binary encoding it is obvious that the cost of one and zero are never same while transmitting data. If different bits can be considered differently then transmission system can become more efficient. Hence, different encoding scheme using unequal bit cost would be necessary. This study proposes an algorithm for data compression where different cost for different bit is considered. Our experiment shows that our algorithm has a better performance than classical Huffman coding in terms of total cost while compressing data. Our variant of Huffman algorithm can give better performance for transmission, storage, speed of execution and so on.**

Keywords—**Huffman algorithm, unequal bit cost, data compression, source coding**

I. Introduction

Few years back hard disks of personal computer were too small. With invention of new technology size of hard disks are getting bigger. Now a days hundreds of gigabytes are stored in a smaller chip. As the size of hard disks are growing bigger, size of the files are also keeping pace with it. So storing all the files easily is still a problem. Compressing the files is the most efficient technique to store optimally [1].

Data compression or source coding uses fewer bits to represent encoding information than the original. Compression can be of two types, lossy compression and lossless compression. In lossless compression there is no data loss after decompression. To be lossless a technique should not be ambiguous. Ambiguity is a situation in which something can be understood in more than one way; i.e. an expression or statement that has more than one meaning [2].

In lossless compression no data is lost after decompression. System can recover the original data from the compressed data in lossless compression but in lossy compression data can't be fully recovered as the technique removes less important or unnecessary bits. The process of reducing the size of a data file is referred to as data compression [3].

Removing all the vowels from a sentence can be considered as a lossy compression technique. For example let a sentence "The earth is round". If we remove all the vowels from this sentence then it becomes "Th rth s rnd".

This shrinks the original 15 characters down to just 9 and requires only 60% of the original space. We can try to checking consonant patterns with English words to decompress, but we cannot reliably reconstruct the original sentence. Is the compressed word "s" an abbreviation for the word"as" or the word"is" or "so?" A human can usually determine it by context, but for a computer it is quite impossible to faithfully reproduce the original. For files containing text, we usually want a lossless scheme so that there is no ambiguity when recreating the original meaning and intent [1]. The algorithm to completely perform Huffman encoding and decoding has too many implementation details, everything that is required is explained in detail in [4].

A method for transmitting information is Morse code. Communication with Morse code happens by on-off tones or clicks that is easily understandable by a skilled receiver without any use of special tool or equipment. It is named for Samuel F. B. Morse, who invented telegraph. Morse code encodes the message as standard sequenced signals called "dots" and "dashes". Dots are short in length and dashes are long. Version of Morse Code exists for those languages which have more than 26 characters. The duration of a dash is three times the duration of a dot. There is a short silence between each dash and dot which is equal to the dot duration. However, the Morse code scheme suffers from the prefix problem [5].

A widely used example of a lossless compression is Huffman coding. This technique assigns variable length code to input characters. Frequencies of the input characters determine the length of the code. Characters which are highly frequent get the smaller code and the characters which are lowly frequent get the larger code. Huffman coding generates a prefix free code. Prefix code is a code system where no code is prefix of another code segment. A prefix free code is uniquely decodable. Huffman coding creates prefix free code from a set of characters and their frequencies. This problem is very well studied and has a well-known $O(n)$ time greedy algorithm [2]. Huffman's coding outputs the fastest letter by letter decipherable encoding of English letters into binary digits. The mean time per letter is $T = 4.120$ units, about half the Morse time [6].

Application of Huffman coding in computer science is very wide. There exist many variants of Huffman coding that compress data. It doesn't give optimal performance when codewords for every encoded characters are of same length. Huffman coding solves this problem by generating variable length code by using probability of occurrence of every character. Still there are some problems in Huffman coding. One of the most common problems is cost of the encoded code are counted by the length of the codeword. Where actually the cost of the individual characters of the encoding alphabet should be different. Because in reality it is not possible that the transmission and storage costs of 1 and 0 are same. Though data compression and transmission system in current time is implemented using same cost for one and zero, in future the data transmission and compression systems will use different cost for different bits and will become more efficient.

II. Scope of The Work

Ambiguity must be avoided in an encoding scheme so that data can be retrieved. To avoid ambiguity in case of fixed number of elements in an alphabet we use predefined code. In case of infinite number of elements this predefined codes become invalid. Our study here is also limited to fixed number of elements in an alphabet like [7].

Several approaches have been considered to improve the compression of data by creating improved versions of Huffman coding. One of them creates a minimum cost monotonic sequence using the monge property and SMAWK algorithm then converts that to lopsided tree [8]. But this approach is only limited to the case when the cost of the elements in an alphabet is positive integers. Golin presented a polynomial-time approximation scheme for the problem that is based on relaxation of Huffman coding with unequal letter costs [9]. He first created a prefix code which is $(1+\epsilon)$ minimal, then converts that to leveled prefix code and then converts that to prefix-code. Another top down approach was introduced in 2014 [7] where Huffman coding for binary encoding was modified by using unequal bit cost. They considered different cost for both bits 0 and 1. Cost of 0 is considered one third than the cost of 1. First the algorithm makes a Huffman tree depending on the total number of input characters. Then it assigns characters to the leaf nodes of that tree. After that it resolves any conflict between the nodes by swapping them. A conflict happens when cost of any node C_i is greater than the cost of any node C_j but the frequency of the node f_i is also greater than f_j.

Our proposed technique also considers unequal cost for bits 0 and 1. As in [7] when it takes a standard amount to transmit One, it takes one third of that to transmit Zero. Application of the proposed algorithm will not be limited to transmission only it can be extended to storage, execution speed and likewise situations.

III. Proposed Algorithm

The proposed algorithm which builds the tree by considering not only the number of the input symbols as [7] rather than frequency of the symbols. We are counting the cost of every node while constructing the tree. Initially every node cost is zero. A parent node cost is determined by the total cost that it's both children are participating and frequency is determined by adding it's both children's frequency. A left child's participation cost is determined by adding that child's frequency and cost. A right child's participation cost is determined by multiplying its frequency with three and then adding with its cost. A parent node is created by taking the node as the right child which giving minimum cost as a right child and then taking the left child which giving minimum cost as a left child except the right child.

Algorithm 1 Considers unequal bit cost for compression

```
    Input: Distinct symbol and their frequencies
    Output: Cost considering tree
 1: Take every input character and their frequency as nodes in a data structure and assign cost zero to every one of them
 2: Set costs of the left and right child of the binary tree
 3: repeat
 4:     pop a node R from the data structure which will give minimum cost as a right child
 5:     pop a node L from the data structure which will give minimum cost as a left child
 6:     make a new node P
 7:     frequency of P = frequency of L + frequency of R
 8:     cost of P = (left child cost*frequency of L + cost of L) + (right child cost*frequency of R + cost of R)
 9:     make L left child of P
10:     make R right child of P
11:     insert P into the data structure
12: until there is more than one entry in the data structure
13: calculate individual cost of every node in the tree
14: repeat
15:     if frequency of any node N1 is greater than frequency of any node N2 and cost of N1 is greater than cost of N2 then
16:         swap N1, N2 and reassign frequency and cost of affected nodes
17:     end if
18: until no node has higher frequency and cost then any other node
```

Let, Left child cost = A

and, Right child cost = B where $A < B$

Initial cost of every node is 0. Initial frequency of every node is input symbol frequency. Participation of cost of both children to a parent node will be calculated throughout following equations.

Left child cost participation $P_L = A * frequency + NodeCost$

Right child cost participation $P_R = B * frequency + NodeCost$

In step 4 and 5 the algorithm will search for the node which will give minimum cost as right child and left child using above equation. Then a new node will be created as their parent node in step 6. Now the cost and frequency of the parent node are counted by the following equations.

Parent node cost = $P_L + P_R$

Parent node frequency = $LeftChildFrequency + RightChildFrequency$

Algorithm 1 shows the process of creating Huffman tree considering unequal cost for different bits, where the left child's will be assigned code Zero and right child's will be assigned code One. Traversing the tree one can get codes for every character.

IV. Algorithm Execution

The algorithm can be considered as a modified version of classical Huffman coding because like classical Huffman coding the algorithm executes in bottom up approach. The classical Huffman coding algorithm compresses data considering frequency of the input characters but not the bits that are used to represent the encoded message. As we are considering both frequency of the input characters and cost of encoding bits our algorithm creates a significant change in the compression of message while unequal cost of the bits is considered.

Algorithm proposed in [7] shows a top down approach considering unequal bits cost where the tree is created based on total number of input characters rather than the frequency of the characters. To eradicate that problem we designed the bottom up approach. For input sequence in Table I the output tree for the proposed algorithm is shown in Fig. 1

Table I
Dataset for algorithm input

Symbol	A	B	C	D
No	50	20	4	2

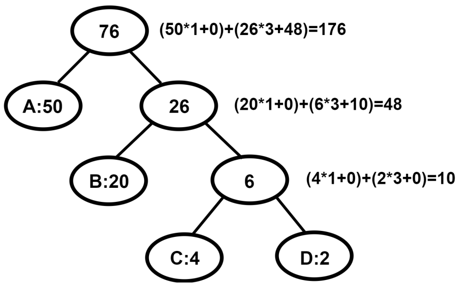

Figure 1. Output tree for proposed algorithm

V. Performance Evaluation

When it comes to the case of variable length coding considering unequal bit costs the system should use less time rather than the costlier ones for better performance. Our algorithm works in that way. It uses less cost for much frequent characters so that the total cost is lesser. To evaluate the performance of our algorithm we will consider the dataset in Table II. Output using classical Huffman algorithm, algorithm in [7] and our proposed algorithm for the dataset in Table II are shown in Table III, Table IV and Table V consecutively.

Table II
Frequency of elements from input

Symbol	A	B	C	D	E	F
No	120	105	80	35	6	4

Table III
Output for classical huffman coding

Symbol	Frequency	Codeword	Cost	Total Cost
A	120	0	1	120
B	105	10	4	420
C	80	111	9	720
D	35	1101	10	350
E	6	11001	11	66
F	4	11000	9	36
				Total Cost=1712

Table IV
Output for algorithm in [7]

Symbol	Frequency	Codeword	Cost	Total Cost
A	120	1	3	360
B	105	0000	4	420
C	80	010	5	400
D	35	001	5	175
E	6	0001	6	36
F	4	011	7	28
				Total Cost=1419

Table V
Output for proposed technique

Symbol	Frequency	Codeword	Cost	Total Cost
A	120	1	3	360
B	105	01	4	420
C	80	000	3	240
D	35	0010	6	210
E	6	00110	9	54
F	4	00111	11	44
				Total Cost=1328

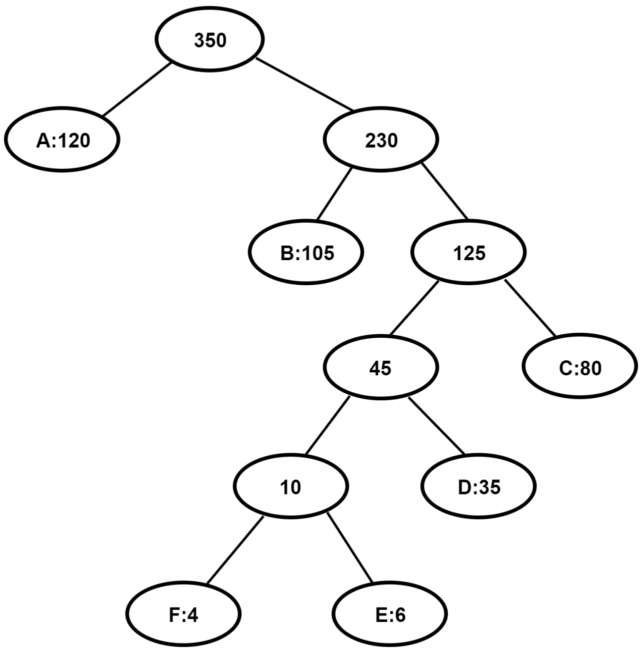

Figure 2. Output tree for Huffman algorithm

Table VI
COMPARATIVE EFFICIENCY OF TWO DIFFERENT SCHEMES

Symbol	Frequency	Codeword for [7]	Proposed Codeword	Cost for [7]	Proposed Cost	TotalCost for[7]	Proposed TotalCost
A	26	00001	0000100	7	9	234	182
B	59	100	001	5	5	295	295
C	5	10101	000101	11	10	50	55
D	14	10100	00011	9	9	126	126
E	92	01	01	4	4	368	368
F	8	10110	000011	11	10	80	88
G	3	10111	0000101	13	11	33	39
H	43	0001	11	6	6	258	258
I	69	001	10	5	4	276	345
J	30	110	000100	7	8	240	210
K	50	00000	000000	5	6	300	250
K	18	111	000001	9	8	144	162
						Total Cost=2404	Total Cost=2378

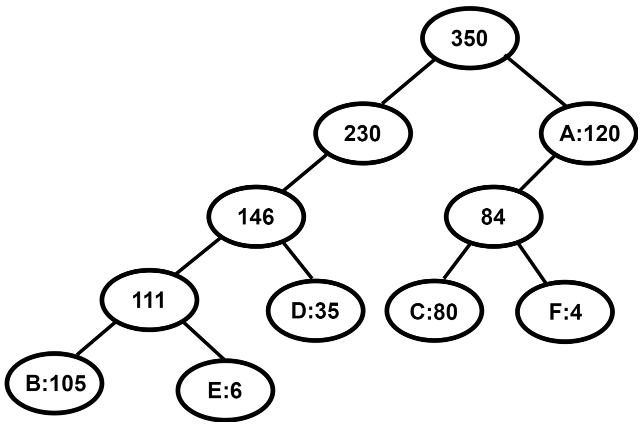

Figure 3. Output tree for algorithm in [7]

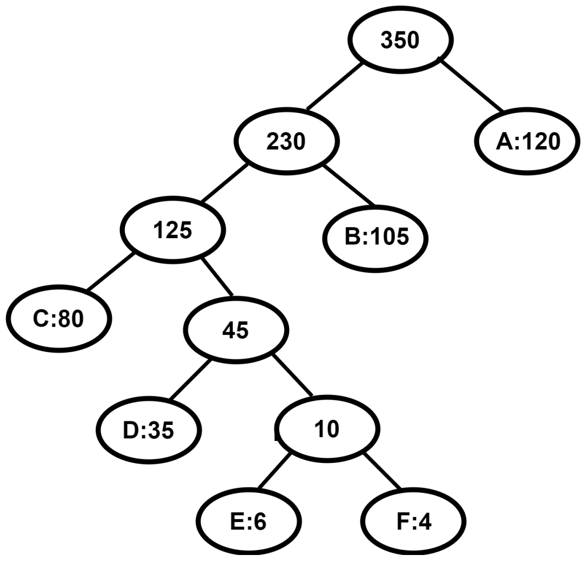

Figure 4. Output tree for proposed algorithm

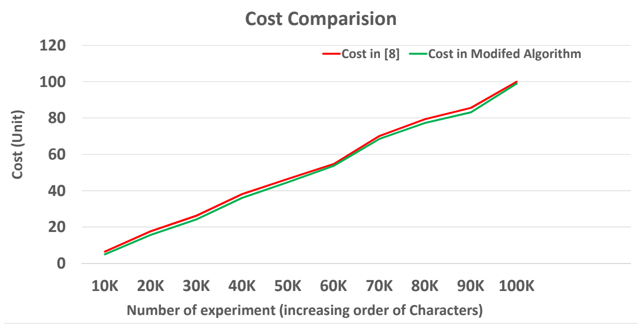

Figure 5. Cost comparison between our technique and algorithm in [7]

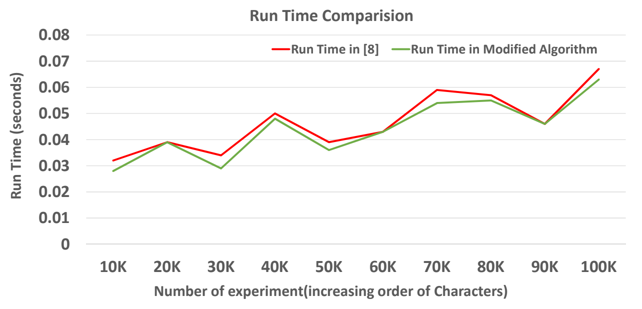

Figure 6. Runtime comparison between our technique and algorithm in [7]

We can see from Table III and Table V that there is a great reduction in cost for our algorithm than classical Huffman algorithm. A comparison between our proposed algorithm and algorithm given in [7] is shown in Table VI for a particular dataset. For twelve characters where total frequency is 417, our algorithm gives total cost 2378 unit and cost for algorithm given in [7] is 2404 unit. So the total cost is reduced applying our algorithm.

Output tree for Huffman algorithm, algorithm in [7] and for our proposed algorithm are shown in Fig. 2, Fig. 3 and Fig. 4.

Although a cost-considering variable-length code provides a cost optimal solution it can sometime use more binary bits than in Huffman code. Several input set is considered for evaluating the proposed algorithm. The statistical aggregated result shows that the output of the proposed algorithm that considers variable length cost is in line with the theory. Our algorithm gives better result in most of the cases. Graph on Fig. 5 and Fig. 6 shows the comparison of cost and runtime between our proposed algorithm and the algorithm given in [7].

VI. Conclusion

The cost of bits for creating the codeword is not considered in a equal letter cost algorithm, it considers the length of the codeword. So the compression ratio is high in this system but cost of the code is also high. As the algorithm assigns low length of encoding string for higher frequency characters and vice versa it does not consider the cost of bits, it is likely that the cost would be much higher. Our proposed algorithm considers the cost of the bits that are creating the encoding string so it assigns least costly codeword to the most frequent symbol. As the most frequent symbols contribute more for decompression process the overall cost is reduced.

Aim of a study is to increase efficiency and reduction of cost in it's particular area. In our case, it is the reduction of cost for compression. It can be applicable in the case of storage or transmission, etc. There may be other schemes, which are more efficient than variable-length Huffman code that considers only compression. One such coding scheme is cost-considering variable-length codeword. Similar to this study, data communication considering unequal bit cost was considered in [10], [11], [12], [13], [14], [15].

We will try to put more effort to give a polynomial time algorithm in future to reduce more cost considering unequal bit costs for variable length codes. We believe that the theories for unequal bit cost will become mature and new system will be built considering this theories. By considering cost-considering code future systems will perform more efficiently.

References

[1] J. Zelenski and K. Schwarz, "Huffman encoding and data compression," *Spring Handout*, vol. 22, 2012.

[2] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[3] E. Britannica, *Encyclopaedia Britannica 2011*. Encyclopaedia Britannica, 2011.

[4] C. Handout, "Data compression and huffman encoding," *space*, vol. 32, p. 00100000.

[5] P. D. Grunwald and P. Vitanyi, "Kolmogorov complexity and information theory," *Journal of Logic, Language and Information*, vol. 12, no. 4, pp. 497–529, 2003.

[6] E. N. Gilbert, "How good is morse code?" *Information and Control*, vol. 14, no. 6, pp. 559–565, 1969.

[7] S. Kabir, T. Azad, A. A. Alam, and M. Kaykobad, "Effects of unequal bit costs on classical huffman codes," in *Computer and Information Technology (ICCIT), 2014 17th International Conference on*. IEEE, 2014, pp. 96–101.

[8] P. Bradford, M. J. Golin, L. L. Larmore, and W. Rytter, "Optimal prefix-free codes for unequal letter costs: Dynamic programming with the monge property," *Journal of Algorithms*, vol. 42, no. 2, pp. 277–303, 2002.

[9] M. J. Golin, C. Kenyon, and N. E. Young, "Huffman coding with unequal letter costs," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 785–791.

[10] N. M. Blachman, "MInimum-cost encoding of information," *Transactions of the IRE Professional Group on Information Theory*, vol. 3, no. 3, pp. 139–149, 1954.

[11] L. E. Stanfel, "Tree structures for optimal searching," *Journal of the ACM (JACM)*, vol. 17, no. 3, pp. 508–517, 1970.

[12] N. Cot, "A linear-time ordering procedure with applications to variable length encoding," in *Proc. 8th Annual Princeton Conference on Information Sciences and Systems*, 1974, pp. 460–463.

[13] D. M. Choy and C. Wong, "Bounds for optimalα-β binary trees," *BIT Numerical Mathematics*, vol. 17, no. 1, pp. 1–15, 1977.

[14] S. Verdu, "On channel capacity per unit cost," *IEEE Transactions on Information Theory*, vol. 36, no. 5, pp. 1019–1030, 1990.

[15] B. Varn, "Optimal variable length codes (arbitrary symbol cost and equal code word probability)," *Information and Control*, vol. 19, no. 4, pp. 289–301, 1971.