Novel bit-level Adaptive and Asymmetric Data Compression Technique

Shashwat Tiwari
School of Computer Science
UPES, Dehradun

Ayushi Lal
School of Computer Science
UPES, Dehradun

Shivani Agarwal
School of Computer Science
UPES, Dehradun

Ayush Kumar
School of Computer Science
UPES, Dehradun

Anupam Singh
School of Computer Science
UPES, Dehradun

Nitin Arora
Electronics & Computer Discipline
IIT Roorkee

ABSTRACT
This paper works on a detailed and per-formative evaluation of a bit-level, adaptive, and asymmetric data compression scheme that is based on the adaptive character word length algorithm. It can be used with statistical compression techniques to increase efficiency. In this mathematical technique, the data word is converted into codewords (Binary form) then the binary coded file is compressed using 8 bits character word length. In this new Algorithm, an optimum character word length b is calculated where (b>8), so that a factor of (b/8) increases the compression ratio. To validate this algorithm, it is used as a complementary with Huffman code to compress a source text file with randomly distributed characters of different frequencies. This scheme is used to compress several text files into smaller bit-level files and can be used to achieve higher competitive skills more than state-of-art tools.

Keywords
Data compression; Coding; Huffman coding; Binary codes.

1. INTRODUCTION
In this modern era of technology, everything is comprised of data. So, the problem in hand is to make the use of an adaptive compression technique, which can improve the compression ratio of existing compression techniques and could enhance the storage capacity of various storage devices by compressing the size of the original text file in a lossless manner. Data Compression Algorithms [1] aim at minimizing the size of the data so that it occupies less amount of disk space and even help in reducing the network congestion since compressed data would use less bandwidth while being transmitted over a data communication channel. Text compression aims at substituting the original symbol with a shorter symbol in the source code, which contains the same information but with a smaller system. It can also be handled at a bit level as each word has its own specific binary representation. Text compression influences all data structures in the secret code. Data compression can be broadly divided into two different categories, namely lossless and lossy. Lossless compression techniques can usually achieve a 2:1 to 8:1 compression ratio. The lossy method provides higher throughput but at the cost of loss in quality. Data communication and data storage applications have benefited greatly from data compression methodology. By reducing the size of the transmitted data, the adequate bandwidth of the communications channel can be increased. The apparent advantage of data storage applications is that smaller data require less storage space. Thus, the proper storage capacity of any storage medium is increased if the data are compressed. There are three significant types of compression techniques: Substitution, Statistical, and Dictionary-based compression. This paper aims at a statistical method, which involves the generation of the shortest average code length based on an estimated probability of the characters. In this paper, one of the analytical data compression techniques known as the Huffman coding is used. The data compression techniques convert 8 bit of data from the original file to the compressed file. It is used as an 8-bit adaptive character word length. In this paper, the proposed algorithm, namely, which is a parallel algorithm for the compression techniques. The compression ratio can be increased by a factor of b/8, where b is the character word length, and b>=8. The coding format yields a low entropy binary sequence so that it grants a higher compression ratio.

2. RELATED WORK
Many researchers work in this area. This section contains the word done by some of the researchers. The data compression techniques can be broadly classified as lossy or lossless data techniques [2] In the lossless data technique as the name suggests the data remains in the compressed file as in the original file [3]. It can be used in executable codes, word processing, and files, etc. In lossy data compression [4], the compressed file loses some of the data from the original file. Especially the redundant data in the code are removed from the compressed file to increase the compression ratio[5]. They are mostly used with video and sound files. The data compression can be of 3 types: Substitution, Statistical, and Dictionary technique [6-7]. Substitution means swapping of repetitive character by the smaller representation. Statistical involves the generation of shorter average code length based on an estimated probability of the characters. The dictionary technique involves the substitution of sub-strings of indices or pointer code, relative to a dictionary of the sub-strings. Huffman coding is a lossless data compression technique. In this way, the characters in a data file are converted to a binary code, where the most popular characters in the file have the shortest binary code, and the least common have the longest[8-10] The first step in building a Huffman code is arranging the characters in the ascending order of their frequencies. The second step is to create a binary tree. The limitation of Huffman is that they are not very good with the randomly distributed data as the frequencies of different characters would be almost the same. When used along with other data compression techniques, they propose a better compression ratio and thus more efficiency. Complexity of the algorithm, computational CPU time, the compression ratio, the amount of the memory required, and error control techniques, the compressional style are the different criteria for comparing. [11-12]

3. PROPOSED ALGORITHM

The proposed algorithm, along with the pseudocode and flow chart for the compression and decompression of the file is discussed in this section.

Proposed Algorithm for the compression of the file is:

Algorithm for file compression

1. Take character file (.txt) as input and read all characters
2. Store individual characters and their frequency in a dynamic array.
3. Create a leaf node for each unique character and build a min-heap of all leaf nodes.
4. Extract two nodes with the minimum frequency from the min-heap.
5. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and another removed node as its right child. Add node to the min-heap.
6. Repeat Step 5 and Step 6 until the heap contains only one node. The remaining node is the root node, and the tree is complete.
7. Store the binary codes in the "huff.txt" file.
8. Open the "huff.txt" file in reading mode and shift the file pointer to the end.
9. Calculate the decimal equivalent of the current binary digit and check if it is less than 127.
10. Add 33 to the decimal value and store it into a new compressed file.
11. Repeat 9 and 10 till the first character of the file is reached.
12. End

Pseudocode for the compression of the file is:

Pseudocode (compression)

1. **INPUT** character file (.txt)
2. f=**open**(".txt",r)
3. ***while*** ch!=NULL

a. Store character-> char array

b. Store Frequency -> frequency array

4. ***end while***
5. **set** char array in increasing order
6. **merge** two smallest frequency nodes
7. ***set*** 0-> smaller node
8. ***set***1-> higher node
9. ***if*** nodes left

a. REPEAT 8,9,10

10. ***end if***
11. ***else***

a. Traverse tree (root to char)

12. ***end else***
13. STORE binary code –> huff.txt
14. F1=open("huff.txt",r)
15. Calculate decimal value
16. ***if*** decimal val<127

a. Then add 33+decimal value

b. Store decimal value -> comp.txt

17. ***end if***
18. ***repeat***19 till the first character
19. ***end***

Proposed Algorithm for the Decompression of file is:

Algorithm for Decompression

1. Open compressed file in reading mode.
2. Read a character from the file.
3. Convert character to its decimal equivalent and subtract 33.
4. Write equivalent binary digits to "intermediate.txt."
5. Repeat steps 2, 3, 4 till the end of the file is reached.
6. Open "Intermediate.txt" in reading mode and move the file pointer to the end.
7. Read a bit.
8. If it matches any Huffman code write the character to the uncompressed file.
9. If it doesn't match, then include more bit.
10. Repeat step 6,7,8 till no more bits are left.
11. Close the uncompressed file and delete all intermediate files.
12. End

Pseudo code for the Decompression of file is:

Pseudo code (decompression)

1. ***START***
2. f=open("comp.txt",r)
3. f.read()
4. int a=ch
5. c=a-33
6. f1=open("intermediate.txt",'w')
7. f1.write(c)
8. ***if*** more characters

a. ***return*** step3

9. ***end if***

10. ***else***

a. f2=open("intermediate.txt",'r')

b. move pointer eof

11. ***end else***

12. Read bits

13. ***if*** bit matches Huffman code

a. F3=open("fileorg.txt," 'w')

b. F3.write(c)

c. ***if*** more characters

i. RETURN step 15

d. ***end if***

e. ***else***

i. f3.close()

f. ***end else***

14. ***end if***

15. ***else***

a. ***RETURN*** step 15

16. ***end else***

17. Delete intermediate files

18. ***END***

4. RESULTS AND DISCUSSION

The proposed algorithm is achieving a coding rate of (2.94×8)/b or a compression ratio of b/2.94, where b is the optimum character word length. Thus, for 9 and 10 character word lengths, for example, the coding rates are 2.16 (C =3.06) and 2.35 (C =3.40), respectively.

The optimum value of b depends on several factors:

- The size and the type of the data file
- The characters frequencies within the data file
- The distribution of characters within the file
- The equivalent binary code used for each character.

$$Compression\ Ratio = \left(\frac{Compressed\ file\ size}{origional\ file\ size}\right) \times 100 \qquad (1)$$

The compression ratio is calculated using eq. 1. Table 1 shows the size of the file using the proposed algorithm and Huffman algorithm using different test cases with different original file size. The graphical representation of the compressed file size is shown in figure 1.

Table1: Original file size and compressed file size using the Huffman Algorithm and proposed algorithm

Test case No.	Original file size (Bytes)	File size after compression in Bytes (Huffman Algorithm)	File size after compression in Bytes (Proposed Algorithm)
Test case 1	73	41	37
Test case 2	94	51	49
Test case 3	159	95	83
Test case 4	243	146	130
Test case 5	315	190	170
Test case 6	436	262	236

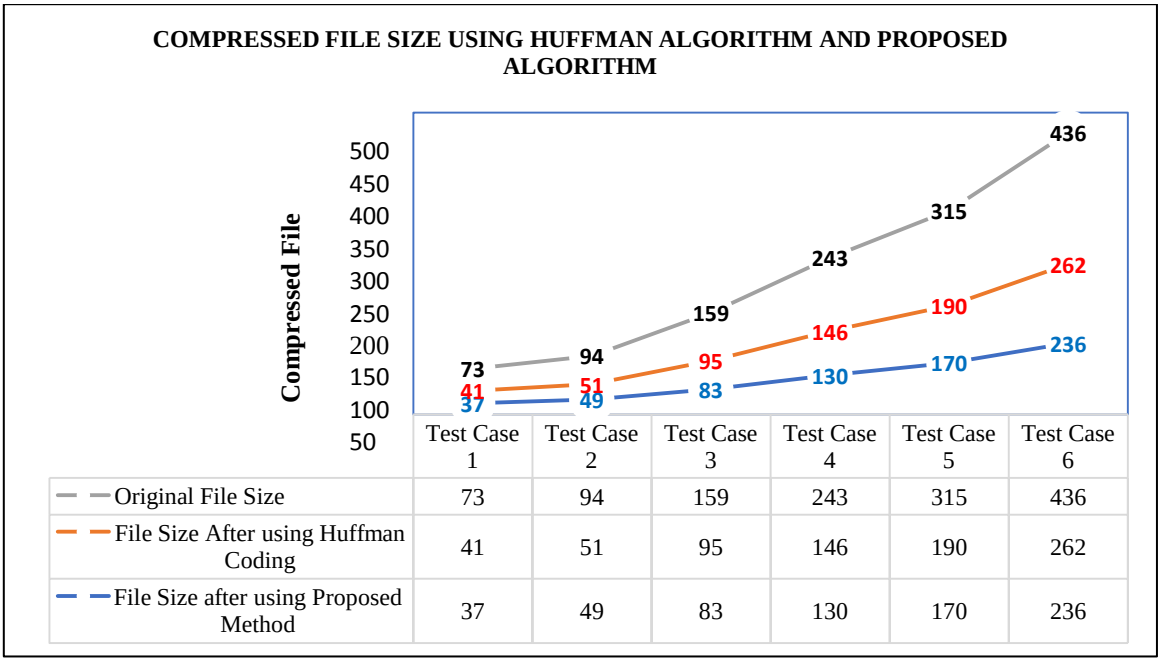

Fig. 1: Compressed file size using the Huffman algorithm and the proposed algorithm

Results show that the proposed algorithm is having a better compression ratio of the Huffman algorithm.

5. CONCLUSION AND FUTURE SCOPE

In this paper, the text file is being converted into a compressed file and the compressed file is converted back to the original file. The proposed algorithm is a lossless compression algorithm and having a better compression ratio than the Huffman algorithm. In the future, more algorithms can be designed with a better compression ratio. The proposed algorithm can be used for reducing network traffic, and it can also be used to enhance the storage capacity of various storage devices.

6. REFERENCES

[1] M.K. Pandya, Data compression: efficiency of varied compression techniques, Formal Report, Brunel University, 2000

[2] www.vectorsite.net/ttdcmp1.html

[3] www.data-compression.com/lossless.shtml

[4] Hussein Al-Bahadilia, Shakir M. Hussainb, An adaptive character wordlength algorithm for data compression, Computers and Mathematics with Applications 55 (2008) 1250–1256

[5] Hussein Al-Bahadili1 Shakir M. Hussain2, A Bit-level Text Compression Scheme Based on the ACW Algorithm, International Journal of Automation and Computing, 7(1), February 2010, 123-131 DOI: 10.1007/s11633-010-0123-6

[6] H. Plantinga. An asymmetric, semi-adaptive text compression algorithm. In Proceedings of IEEE Data Compression Conference, 1994.

[7] J. Adiego, P. de la Feunte. On the use of words as source alphabet symbols in PPM. In Proceedings of Data Compression Conference, IEEE, pp. 435, 2006.

[8] S. Nofal. Bit-level text compression. In Proceedings of the 1st International Conference on Digital Communications and Computer Applications, Irbid, Jordan, pp. 486–488, 2007.

[9] Arora, N., Tamta, V., and Kumar S., 2012. A Novel Sorting Algorithm and Comparison with Bubble Sort and Selection Sort. International Journal of Computer Applications. Vol 45. No 1. 31-32

[10] Arora, Nitin &Kumar Tamta, Vivek & Kumar, Suresh. (2012). Modified non-recursive algorithm for reconstructing a binary tree. International Journal of Computer Applications. 43. 25-28. 10.5120/6141-8386.

[11] Arora, N., Kaushik, P.K., Kumar, S.: Iterative method for recreating a binary Tree from its traversals. International Journal of Computer Applications 57(11), 6–13 (2012).

[12] Garima Pandey, Nitin Arora, Mamta Martolia. "A Novel String Matching Algorithm and Comparison with KMP Algorithm" in International Journal of Computer Applications (0975 – 8887) Volume 179 – No.3, December 2017.