Symbiosis

Research Article | Journal of Computer Science Applications and Information Technology | Open Access

Received: January 07, 2017
Accepted: January 31, 2017
Published: February 15, 2017

An Improved Decoding Technique for Efficient Huffman Coding

Jamil As-ad[1], Mohibur Rahaman[2], Rashed Mustafa[3*], Zinnia Sultana[4] and Lutfun Nahar[5]

[2,3,5]Department of Computer Science & Engineering, University of Chittagong, Bangladesh
[1,4]Department of Computer Science & Engineering, International Islamic University Chittagong, Bangladesh

***Corresponding author:** Rashed Mustafa, Associate Professor, Department of Computer Science & Engineering, University of Chittagong, Chittagong-4331, Bangladesh, Tel: 8801819646333; E-mail: rashed.m@cu.ac.bd

1 Abstract

In today's world storing numerous data or information efficiently is a significant issue because of limited storage. Every moment we need to transfer large volume of information soundly and correctly. A storage space/memory storage required to store this information is. The way of communication which is also constricted with confined communication lines. These incompetence drives to need of data compression. Compression makes a reduction in memory storage, data transmission time, communication bandwidth and the ultimate cost savings also. To compress data efficiently and effectively one of the most popular and widely used techniques is Huffman compression. It is a lossless compression technique that enables the restoration of a file to its authentic/key state, having not to loss of a single bit of data when the file is uncompressed. It will be more efficient by reducing the memory requirements for Huffman tree. This article aimed at reducing the tree size of Huffman coding and also explored a newly memory efficient technique to store Huffman tree. As a consequence we also designed an encoding and decoding algorithm. Our proposed technique is much more efficient than most of the/all other existing techniques where to represent the Huffman tree structure total memory requirements are 9n-2 bits for the worst case, average and also for the best case. The results obtained significant improvements over previous/existing work.

2 Keywords:

Tree Compression; Static Huffman Code; Data Structures; Complete Binary Tree;

3 Introduction

Human world is now undergoing the most intense technological revolution. We are still living in the era of information. Every instance of time we have to maintain a large volume of data or information and store them as well. But still storage space is considered to be limited for solving our purpose. To work out in storing data or information in an efficient manner is an important factor/issue. For this reason, a technique, namely compression, is used to act perfectly using the limited storage space.

Compression is accomplished by using algorithms or formulas which describes the shrinking procedure of the size of data [10, 11]. The prime performance metric of data compression is compression ratio [12]. Data compression can be lossless or loss. Lossless compression occupies the restitution of a file to its exact status, without having loss of a single bit of data when the file is uncompressed. For example- text compression, spreadsheet files etc. are lossless technique as because loss of a single character, numbers, or words may convert a message into completely different meaning.

The key objective of this research is to invent a technique to compress the tree size of Huffman compression by a memory efficient representation of Huffman tree [8, 9]. Significant improvements of the tree compression ratio will enhance the data compression ratio. Then designing encoding and decoding algorithm for new technique so that encoding and decoding is accomplished with minimal effort.

In this paper, we've proposed a new encoding-decoding technique to compress the Huffman tree size in an efficient manner

Citation: As-ad J, Rahaman M, Mustafa R, Sultana z, Nahar L (2017) An Improved Decoding Technique for Efficient Huffman Coding. J Comp Sci Appl Inform Technol. 2(1):1-5

and compared the performances (with respect to compression ratio, savings bits etc.) with the previous works. We also have investigated the limitations of some previous works. The proposed method is also a bit representation technique.

The organization of residual of the paper is as follows. Literature Review is explained in section 4. Methodology is described in section 5. Comparison and Results are discussed in the section 6-7. Limitations and scope of future work are depicted in the next section. Finally, we conclude the paper in section 9.

4 Literature Review

Includingoriginal Huffman coding most of the previous works are based on byte representation of Huffman tree. Butinreference [1], the author zinnia et al. proposed the bit representation of Huffman tree, where the authors use some terminology namely the conception of "circular leaf node", "node with 2 internal son nodes" and give new concepts of "upper leaf node", "external limb", "internal limb", "antenna limb" [1] to solve critical limitations of existing works and they were promising to solve some existing problems.

In reference [2], the authors Choudhury and Kaykobad proposed a new data structure for Huffman tree representation in which in addition to sending symbol codes, codeword for all the circular leaf nodes are sent. They decoded the text by using the memory efficient data structure proposed by Chen et al.

In reference [5], the authors Bei Chen, Hong-Cai Zhang, Wen-Lun Cao, Jian-Hu Feng introduced a new Huffman coding method based on number characters. The traditional 256 code table is replaced by the 0-9 characters, the space character and the enter character in this method. In reference [4] the authors, Wang, S.S-W. Wu, J.J.-L. Chuang, S.S.-C. Hsiao, C.C.-C. Tung, Y.Y.-S. proposed a Lagrangian multiplier based penalty resource metric to be the targeting cost function and addresses the optimization problem of minimizing the number of memory access subject to a rate constraint for any Huffman decoding.

To lessen the memory size and fix the process of searching a symbol in a Huffman tree, Pi Chung Wang et al. proposed a memory efficient data structure to represent the Huffman tree utilizing the property of the encoded symbols, which uses memory **nd** bits, where **n** is the number of source symbols and **d** is the depth of the Huffman tree. Using single-side growing Huffman tree, based on the proposed data structure, an O (log n)-time Huffman decoding algorithm is introduced [3, 6, 7].

5 Proposed Method

Lots of applications of Huffman encoding method rely on ASCII codes. ASCII is 8-bit character coding scheme. But it is defined only for 0-127 codes which means it can be fit into 7-bits. The leftmost bit of an ASCII coded character is always set to zero i.e. the MSB is guaranteed to be 0. We've taken advantage of that bit.

In our proposed method the whole Huffman tree is traversed in a depth first fashion. When a leaf containing a character is to be saved, we save 7 bits of that character omitting the MSB. When returning back to the parent from a child node the relative position of the child is saved this takes one bit for each edge of the tree.

5.1 Rules for encoding

a. Rule1: When saving a character the MSB of 8 bits is omitted hence 7 rightmost bits are saved.

b. Rule 2: when returning back from a child node to its parent whether it's a leaf or an internal node a single bit containing either 0 if the child is the left child of the parent or 1 if the child is the right child of the parent is saved.

5.2 Proposed encoding algorithm

1. Traverse the tree in a depth first fashion.
2. Repeat steps i and ii until the traversal is not finished

i. If a leaf is encountered save the character in that position according to rule 1.

ii. When returning back from a node to its parent node save its relative position according to rule 2.

5.3 Proposed decoding algorithm

Input: bit stream

Output: Huffman tree

Decoding starts from the left of the bit stream.

Initialization of Current_subtree, subtree stack

Read_character () reads consecutive 7 bits and converts these into an 8 bit ASCII code adding 0 to its MSB.

Read_position () reads a single bit.

Decode_and_build_tree ()

1. Read_character()
2. Current_subtree:= character read
3. **while** scanned bit≠NULL**do**
4. **if** Current_subtree ≠NULL**then**
5. Read_position()
6. **if** position = 0 **then**
7. Current_subtree is a left child. Push it into the stack.
8. Current_subtree := NULL
9. **else**
10. Pop topmost subtree from the stack
11. Create a new node with left child := popped out subtree
12. right child:=Current_subtree
13. Current_subtree := newly created node
14. **end if**
15. **else**
16. Read_character()
17. Current_subtree := character read
18. **end if**
19. **end while**
20. Current_subtree is the required Huffman tree.

Here we depict an example of implementation for clear understanding.

This tree (Figure 1, 2) will be encoded into the following bit sequence. Here every character consists of 7 bits. This representation has been used for easy understanding.

6 Complexity analysis

If there are n number of different characters to be encoded, total number of edges in Huffman tree would be (2*n - 2). In our proposed method every edge is denoted by either 0 or 1 depending

Citation: As-ad J, Rahaman M, Mustafa R, Sultana z, Nahar L (2017) An Improved Decoding Technique for Efficient Huffman Coding. J Comp Sci Appl Inform Technol. 2(1):1-5

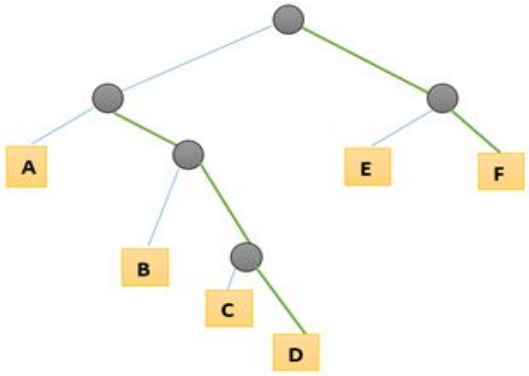

Figure1: Example Tree for Encoding

A	0	B	0	C	0	D	1	1	1	0	E	0	F	1	1

Figure 2: Encoded Bit Stream for Tree of Figure 1

on their relative position i.e. 1 bit per edge. So number of bits to encode the edges will remain same i.e. (2*n - 2) for every case.

Besides, every ASCII coded character comprises of 8 bits where the MSB is 0 for each character. We are ignoring this bit while encoding so that we will require 7 bits for representing each character in our generated encoded bit sequence (explained in Figure 3, 4).

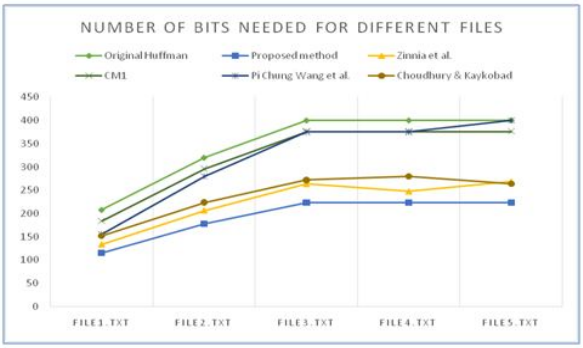

Figure 3: Line Diagram Showing Required Number of Bits for Different Methods

So total space needed would be: (7*n + 2*n - 2) = 9*n - 2 bits

*Authors at reference [2], didn't mention the limitation but this analysis won't work if the tree depth exceeds 7th level (starting from level 0). Then a massive change in algorithm would be needed in that case.

**In reference [1], the best case will be occurred when only the right most branch of the tree is expanded which is not a very frequent occurrence. In other cases redundant bits for saving the external limb and each upper leaf node will be required.

7 Experimental Results

We have encoded the following files with our proposed method (Table 1):

1. file1.txt="Huffman Coding Huffman Coding"
2. file2.txt="A new approach of memory efficient Huffman tree representation"
3. file3.txt="Department of CSE of International Islamic University Chittagong"
4. file4.txt ="I've implemented my proposed algorithm using programming language C because I like it most among all programming languages"
5. file5.txt="Best case complexity occurs when only 1 circular leaf node is considered or only one external limb is considered and all symbols except right brother leaf are the left leafs of the external limb"

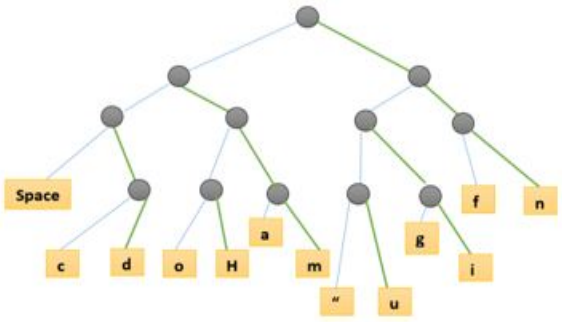

Figure 4: Huffman tree for file1.txt of "Huffman Coding Huffman Coding". Blue colored edges are represented with 0 and green colored edges with 1.

Table 1: Comparison of Space Efficiency with Different Methods

Methods	Memory Requirement	Spaced savings of proposed method compared with existing method
Original Huffman Coding	2n bytes	(2n*8)-(9n-2) = 7n+2 bits
CM1	2n-3 bytes	(2n-3)*8-(9n-2) =7n-22 bits
Method of Pi-Chung Wang; Yuan-Rung Yang; Chun-Liang Lee; Hung-Yi Chang	nd bits	$n(n-1)-(9n-2) = =n^2-10n+2$ bits (approximated)
Choudhury and Kaykobad Method	3n/2 bytes at worst case*	(3n/2)*8-(9n-2) = 3n-2bits (approximated)
Zinnia et al. Method	10.75n-3 bits at worst case**	(10.75n-3)- (9n-2) =1.75n-1 bits (approximated)
Proposed Method	9n-2 bits at worst case	

The following are the comparison with different methods for above files (Table 2,3)

- Original file size is 31 bytes=248 bits
- Total no. of bits required only for compressed text is 113 bits.
- Total no. of bits required only for tree is 2*13 bytes=26*8 bits=208 bits in original Huffman coding.
- Total no. of bits required only for tree is ((13*8) +30) bits=**134 bits in zinnia et al. method.**
- Total no. of bits required only for tree is ((9*8) -2) bits=**115 bits in proposed method.**
- Total size of compressed file along with tree in original Huff-

Citation: As-ad J, Rahaman M, Mustafa R, Sultana z, Nahar L (2017) An Improved Decoding Technique for Efficient Huffman Coding. J Comp Sci Appl Inform Technol. 2(1):1-5

Table 2: Comparison Table of Huffman Tree Compression Efficiency

File Name	Number of Symbols	Number of bits needed to save the Huffman tree					
		Original Huffman	Proposed Method	Zinnia et al. Method	CM1 method	Pi Chung Wang et al. method	Choudhury and Kaykobad method
file1.txt	13	208	115	134	184	156	152
file2.txt	20	320	178	207	296	280	224
file3.txt	25	400	223	264	376	375	272
file4.txt	25	400	223	248	376	375	280
file5.txt	25	400	223	269	376	400	264

Table 3: Comparison table of compression efficiency for file1.txt

File name	Number of bits required (negative value denotes increment of original file size)												
	Original file Size	Original Huffman	Space Savings	Proposed method	Space savings	Zinnia et al. method	Space savings	Choudhury &Kaykobad method	Space savings	Pi Chung Wang et al. method	Space savings	CM1 method	Space savings
file1.txt	248	321	-73	228	20	247	1	265	-17	269	-21	297	-49
Compression Ratio (With respect to original Huffman)		0 %		28.97 %		23.05 %		17.45 %		16.20 %		7.48 %	

man Coding is (208+113) bits=321 bits.

• Total size of compressed file along with tree **in zinnia et al. method is (134+113) bits=247 bits.**

• Total size of compressed file along with tree **in Proposed method is (115+113) bits=228 bits.**

• Tree compression ratio is ((208 - 134) / 208) *100 = **35.58% in zinnia et al. method**

• Tree compression ratio is ((208 - 115) / 208) *100 = **44.712% in proposed method**

• Total file compression ratio **in zinnia et al. method is ((248-247) / 248) * 100 = 0.403**%

• Total file compression ratio **in proposed method is ((248-228) / 248) * 100 = 0.806%**

• Total file compression ratio in original Huffman coding is (248-321)/248*100 = -22.74% = Nil.

8 Limitations and Scope of Future Work

As we have compressed each character in 7 bits by omitting the MSB of 8bit scheme, our algorithm will fit for only working with ASCII characters and will not work properly for extended ASCII character set which uses the MSB also to encode special characters. But efficient mapping might overcome this situation as we don't use all these 256 combinations at once frequently. Another future plan is to get rid of some redundant bits that are used to join the right sub-trees with their left counterparts. Nearly N bits can be saved in best case by efficiently handling this.

9 Conclusion

This paper concentrate on diminishing space and action of the operation because of confined communication line through which data is to be sent. The key purpose of this paper is to reduce the space needed to represent the Huffman tree as it is necessarily has to be sent along with the encoded text to decode successfully. The method proposed here is unique from previously proposed methods which ensure an easy and fast implementation. It outperformed over all the existing methods both in space complexity analysis and experimental results as it takes only 9N - 2 bits to represent the Huffman tree.

10 References

1. Sultana Z and Akter S. A New Approach of a Memory Efficient Huffman Tree Representation Technique. IEEE International conference(ICIEV). 2012. DOI: 10.1109/ICIEV.2012.6317482

2. Rezaul C, Kaykobad M, Irwin K. An efficient Decoding Technique for Huffman Codes. Inform Process Letter. 2002;81(6):305-308.

3. Pi-Chung W, Yuan-Rung Y, Chun-Liang L, Hung-Yi C. A memory efficient Huffman decoding algorithm. Advanced Information Networking and Applications. 2005;2:475-479. DOI: 10.1109/AINA.2005.33

4. Sung-Wen W, Shang-Chih C, Chih-Chieh H, Yi-Shin T, Ja-ling W. An Efficient Memory Construction Scheme for an Arbitrary Side Growing Huffman Table. IEEE International Conference on Multimedia and Expo. 2006:141-144. DOI: 10.1109/ICME.2006.262589

5. Bei C, Hong-Cai Z, Wen-Lun C, Jian-Hu F. Huffman Coding Method Based on Number Character. International Conference on Machine Learning and Cybernetics, 2007;4:2296-2299. DOI: 10.1109/ICMLC.2007.4370528

6. Sung-Wen W, Ja-Ling W, Shang-Chih C, Chih-Chieh H, Yi-Shin T. Memory Efficient Hierarchical Lookup Tables for Mass Arbitrary - Side Growing Huffman Trees Decoding. IEEE Transactions on Circuits and Systems for Video Technology. 2008:18(10):1335-1346. DOI: 10.1109/TCSVT.2008.920968

7. Seymour L. Theory and problems of Data Structures. Mcgraw-Hill Book Company. 2002:249-255.

8.Hashemian R. Memory efficient and high speed search Huffman coding. IEEE Trans Comm. 1995;43(10):2576-2581. DOI: 10.1109/26.469442

9. Yuh-he C, Ja-ling W. An SGH-Tree Based Huffman Decoding. Fourth International Conference on Information, Communications and Signal Processing. 2003;3:1483-1487. DOI: 10.1109/ICICS.2003.1292713

Citation: As-ad J, Rahaman M, Mustafa R, Sultana z, Nahar L (2017) An Improved Decoding Technique for Efficient Huffman Coding. J Comp Sci Appl Inform Technol. 2(1):1-5

10. Huffman DA. A method for construction of minimum redundancy codes. Proceedings of the IRE. 1952;40(9):1098-1101.

11. Schack R. The length of a typical Huffman codeword. IEEE Transactions on Information Theory. 1994;40(4):1246-1247. DOI: 10.1109/18.335944

12. Hong-Chung C, Yue-Li W, Yu-Feng L. A memory- efficient and fast Huffman decoding algorithm. Information Processing Letter. 1999;69(3):119-122.

13. Jenny LJ, Yulia R, Patricia M. Natural Language Translator Correctness Prediction. JCSAIT. 2016;1(1):1-11. DOI: http://dx.doi.org/10.15226/2474-9257/1/1/00107

Citation: As-ad J, Rahaman M, Mustafa R, Sultana z, Nahar L (2017) An Improved Decoding Technique for Efficient Huffman Coding. J Comp Sci Appl Inform Technol. 2(1):1-5