

CONDENSED HUFFMAN CODING, A NEW EFFICIENT DECODING TECHNIQUE

Reza Hashemian, Life Member, IEEE

Northern Illinois University
DeKalb, Illinois 60115, USA

ABSTRACT

A new technique is proposed for decoding Huffman codes. In this technique a Condensed Huffman Table (CHT) for decoding purposes replaces a typical Huffman table. It is shown that a CHT is much smaller in size and the decoding becomes significantly faster. In an example with a typical Huffman table containing 108 codewords, it is shown that a CHT with only 14 codewords is sufficient to perform the decoding.

1. INTRODUCTION

Huffman coding [1] has been shown to be one of the most efficient and simple variable-length coding techniques used in high speed data compression applications. In fact, as it is widely practiced, the combination of Huffman coding and run-length coding provide a near optimal entropy coding technique.

Huffman coding, normally presented by a binary tree, becomes progressively *sparse* as it grows from the root. This sparsity usually causes tremendous waste of memory space, unless a properly structured technique is adopted to efficiently allocate the symbols in the memory[2-7]. In addition, this sparsity may also result in a lengthy search procedure for locating a symbol, hence adding to the delay. There have been a number of techniques developed and reported in the literature [3-13] to reduce the memory requirement and increase the speed for the search.

To directly address the problem, one should realize that there are normally two distinct operations in Huffman coding: codelength determination, and code representation. The first operation is almost unique (although the result may differ from case to case) in every technique, and it was originally established by D.A. Huffman[1]. However, it is the coding representation operation that differs from one technique to another. And it is, in fact, this feature of the encoding that affects both space requirement and the time for search[3]. A common property in any Huffman encoding is that each code must be unique such that it could be recognizable in the input bit-stream with no guard-bit(s) assigned to separate between the two consecutive symbols. As a result, the decoding procedure must recognize the codelength as well as the symbol itself. Another property of the Huffman coding is that, within certain codes that have the same codelength any reordering of the codes in the Huffman table does not change the coding effectiveness (entropy), as long as the change is uniquely adopted by

both, the encoder and the decoder. Nevertheless, the difference may appear in terms of the memory requirement as well as the time for search.

The method presented in this article is based on the type of the Huffman code representation that is reported in [3], called "Single-Side Grown Huffman Table (SGHT)". Due to some unique features in this representation, it is shown that a much smaller table of codes, extracted from a SGHT, can replace the Huffman Table without losing any significant information. Here is a brief procedural description:

1. Encoder generates a SGHT for a set of given symbols and their occurrences (probabilities), and orders the symbols in a symbol-list according to the SGHT. Next, a Condensed Huffman Table (CHT) is extracted from the SGHT.

2. The CHT, the symbol-list, and the smallest codelength are the items that are sent to the decoder. This enables the decoder to extract the symbols from an input bit stream with no need of the Huffman table.

The remaining part of this article is organized in the following sections. In Section 2, we introduce the Single-Side Growing Huffman Table, and how to construct it from a given Table of Codelengths. In Section 3, the Condensed Huffman Table is discussed and generation of the CHT from a SGHT is formulated. In Section 4, an example, derived from a transformed image, is depicted. Here a comparison between a normal Huffman table procedure and a decoding procedure that uses CHT is performed. Finally, Section 5 contains the final remarks and conclusion.

2. SINGLE-SIDE GROWING HUFFMAN TABLE

As reported in [3], a SGHT is a particular Huffman table, which is constructed from a Table of Codelengths (TOCL). As shown in Table 1, a TOCL provides the codelengths of the ordered symbols, s_1, s_2, \dots, s_{18} , as described in [1]. Algorithm 1 formulates a procedure to construct a SGHT from a TOCL.

Table 1 – The Table of Codelengths

| Codelength | Symbol |
|------------|--|
| 2 | s_1, s_2, s_3 |
| 3 | s_4 |
| 6 | s_5, s_6, s_7 |
| 7 | $s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$ |
| 8 | s_{17}, s_{18} |

Algorithm 1: We start from one symbol, say s_1 , with the shortest codelength (if there is more than one symbol with the same codelength any order of such symbols is acceptable). We assign a codeword with "all zeros" and codelength 2 to s_1 . Next, we increment this codeword and assign it to the next symbol in the list with equal codelengths. We continue likewise to cover all symbols with identical codelengths. At this point, we increment the last codeword and move to the symbols in the next higher level and append enough zeros to the codeword in order to adjust its codelength. The rest of the process is to repeat these two operations until we get to the last symbol (s_{18}), which always contains "all ones". To put it in perspective, in general we have:

$$c_i = 00 \dots 0,$$

$$c_{i+1} = (c_i + 1) * 2^{q-p},$$

And

$$c_n = 11 \dots 1,$$

Where, p and q are the code-lengths for the symbols s_i and s_{i+1} , respectively. Note that the initial and the final codewords have unique forms that can be used to check for the errors that may have occurred in the procedure. Table 2 shows the SGHT developed from the TOCL shown in Table 1, and Fig. 1 is its corresponding Huffman tree.

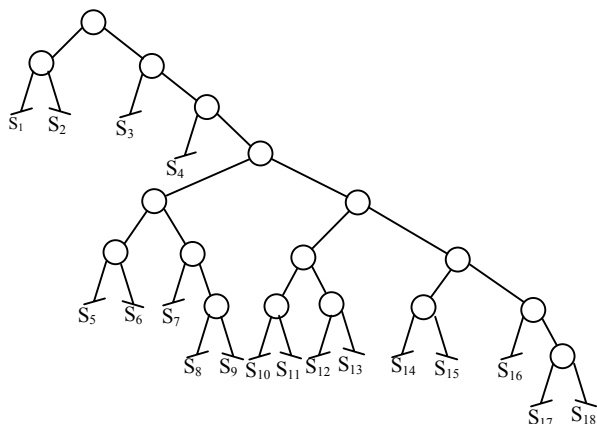


Fig.1- The Single-Side Huffman Tree, from Table 1

Table 2 – The Single-Side Grown Huffman Table

| Symbols | Codewords |
|----------|-----------|
| s_1 | 00 |
| s_2 | 01 |
| s_3 | 10 |
| s_4 | 110 |
| s_5 | 111000 |
| s_6 | 111001 |
| s_7 | 111010 |
| s_8 | 1110110 |
| s_9 | 1110111 |
| s_{10} | 1111000 |
| s_{11} | 1111001 |
| s_{12} | 1111010 |

| | |
|----------|----------|
| s_{13} | 1111011 |
| s_{14} | 1111100 |
| s_{15} | 1111101 |
| s_{16} | 1111110 |
| s_{17} | 11111110 |
| s_{18} | 11111111 |

3. CONDENSED HUFFMAN TABLE

It is shown that a Condensed Huffman Table (CHT), generated from a SGHT, is nearly in the order of magnitude smaller than the size of the original SGHT, whereas it carries the entire information of the SGHT. In fact, a CHT has only one row for each level, except for the smallest level that represents the symbols with the highest probabilities of occurrences.

3.1 Construction of a CHT

To construct a CHT, as pointed out earlier, we will assign one row in the CHT to each level (codelength), except for the smallest codelength, which has none. Basically, if there are multiple codes in the SGHT with the same codelength, only the first one with the smallest codeword value is recorded in the CHT, and each row in CHT contains i) the codelength L_i , ii) a number n_i representing the order of the corresponding symbol in the symbol-list, and iii) the augmented codeword, C_i ; where, the symbol-list is an ordered list of symbols corresponding to the SGHT, and each augmented codeword in CHT is obtained from its counterpart in the SGHT by appending enough zeros to have a maximum codelength L_m .

3.2 Decoding Procedure

To decode an input bit-stream by using the corresponding CHT, we need to go through the following procedure:

1. Take L_m bits from the bit-stream and call it W , and compare W to C_1 , the first (smallest) augmented codeword in the CHT.
2. If W is less than or equal to C_1 then the first L_0 bits in W represent the codeword, c_{0j} , for the intended symbol, where L_0 is the smallest codelength. Also c_{0j} is the address of the desire symbol in the symbol-list (to avoid 0 address we add 1 to the address). Finally, return the excess, $L_m - L_0$ bits into the bit-stream.
3. If, on the other hand, W is greater than C_1 , continue searching for C_i , where C_i is the first augmented codeword in the CHT that is greater than W . Then, the pertinent codelength is L_{i-1} and the codeword is the first L_{i-1} bits in W . Return the remainder, $L_m - L_{i-1}$, bits of W into the input bit-stream for the next process. To find the address of the symbol in the symbol-list, i.e. $n_{i-1,j}$, first find $D = W - C_{i-1}$, and take the first L_{i-1} bits in D , denoted by m_{i-1} . The address is given by $n_{i-1,j} = m_{i-1} + n_{i-1}$.

4. In case W is greater than C_1 and also greater than all other augmented codewords in the CHT, including the last one C_m , then W must itself be the intended codeword. Find $m_m = W - C_m$, and calculate the address of the symbol in the symbol-list, which is $m_m + n_m$.

Notice that any bit stream, although it may not generate the correct output, is decodable in this process. In other words, for any W there is always a codeword in the original SGHT that is identical to the first L_i bits in W , where L_i can take any value from L_0 to L_m . This is, of course, a serious concern in any Huffman coding, and it requires an effective error checking technique, which is beyond the scope of this study.

Example: Take the CHT for the SGHT shown in Table 2, which is given in table 3.

Table 3 - The CHT for the SGHT shown in Table 2

| | Augmented Codeword C | Codelength L | Symbol Position n |
|---|----------------------|--------------|-------------------|
| 1 | c0 | 3 | 4 |
| 2 | e0 | 6 | 5 |
| 3 | ec | 7 | 8 |
| 4 | fe | 8 | 17 |

Now, suppose we have extracted $W = 97$ from the bit-stream. W is smaller than $c0$, and according to the procedure, and with $L_0 = 2$, we take 2 MSBs of W , which is 01 and return the rest to the bit-stream. Then 01 is the codeword and $01+1=2$ is the address of the symbol (s_2). Next let us take $W = f5$ from the bit-stream. The next larger codeword is fe , and prior to that the codeword is ec . With $L_3 = 7$ the pertinent codeword is then 7 bits of W , 1111010, $D = f5 - ec = 09$. Now we take the 7 MSBs of D as 04 and add this to $n_3 = 8$ to get 12, and the symbol is s_{12} . Finally, let us take $W = ff$, which is higher than any codeword in the CHT. By applying the procedure, the intended codeword is $ff = 11111111$, and we add $m_4 = W - C_4$, i.e., $ff - fe = 1$ to $n_4 = 17$ to get 18. The symbol is s_{18} .

4. EXAMPLE FOR AN IMAGE DECODER

Table 4 shows the TOCL for a transformed (DCT and quantized) image (Barb). The number of symbols generated in this transformation turns out to be equal to 108, and with the same number of rows in the SGHT. On the other hand, it is shown that the CHT corresponding to this SGHT has only 14 rows, as shown in Table 5.

Table 4- The Table of Codelengths

| Codelengths | Number of symbols |
|-------------|-------------------|
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |

| | |
|----|----|
| 5 | 4 |
| 6 | 4 |
| 7 | 8 |
| 8 | 13 |
| 9 | 16 |
| 10 | 35 |
| 11 | 17 |
| 12 | 1 |
| 13 | 1 |
| 14 | 1 |
| 15 | 1 |
| 16 | 2 |

Table 5 - The Table of Condensed Huffman Table

| | Augmented Codeword C | Codelength L | Symbol Position n |
|----|----------------------|--------------|-------------------|
| 1 | 4000 | 3 | 2 |
| 2 | 8000 | 4 | 4 |
| 3 | a000 | 5 | 6 |
| 4 | c000 | 6 | 10 |
| 5 | d000 | 7 | 14 |
| 6 | e000 | 8 | 22 |
| 7 | ed00 | 9 | 35 |
| 8 | f500 | 10 | 51 |
| 9 | fdc0 | 11 | 86 |
| 10 | ffe0 | 12 | 103 |
| 11 | fff0 | 13 | 104 |
| 12 | fff8 | 14 | 105 |
| 13 | fffc | 15 | 106 |
| 14 | fffe | 16 | 107 |

Next, we proceed with the decoding procedure. The following list contains the output of a simulation program that takes chunks of data (W) from an input bit-stream and processes them through the CHT. Also the corresponding responses, i.e., the codewords, the codelengths, and the addresses in the symbol-list, are listed in the output.

1- Input in Hex number W : 3791

The Codeword: 0
 The Codelength: 2
 The address in the symbol-list: 1

2- Input in Hex number W : 8000

The Codeword: 8
 The Codelength: 4
 The address in the symbol-list: 4

3- Input in Hex number W : ff37

The Codeword: 7f9
 The Codelength: 11
 The address in the symbol-list: 97

4- Input in Hex number W: ffff

The Codeword: ffff
The Codelength: 16
The address in the symbol-list: 108

5- Input in Hex number W: 5a4e7

The input is too large and is truncated to: b49c
The Codeword: 16
The Codelength: 5
The address in the symbol-list: 8

As expected the process is quite efficient in both memory requirement and delay in decoding and extracting the symbol. To show this, we start looking at the memory being used in our example and the time spent for searching a symbol.

| | |
|--------|--|
| Memory | The symbol-list = 2×108 bytes |
| | The CHT plus $L_0 = 3 \times 14 + 1$ bytes |
| | Total 259 bytes. |

Maximum time-delay: two add/subtract + 13 search.

Note that, due to the high probability of getting short codewords in an input bit-stream, there is often no need for any add/subtract in the decoding process, and the search will also end in the first step. For example, s_1 , s_2 , and s_3 symbols in our first example take about 65% probability of occurrence in the input stream where as the remainder of the symbols statistically occurred only 35% of the time. Therefore, for 65% of the time we need no add/subtract to perform and the symbol is found in the first try.

5. CONCLUSION

A new procedure is adopted for decoding Huffman codes. This procedure is based on a Condensed Huffman Table instead of the original one, and it is shown that the CHT is significantly smaller than the Huffman Table. In addition the procedure is faster in searching for the codeword and its corresponding symbol in the symbol-list. An example from an image decoding is worked out, which reviles both saving in memory and time for decoding processes.

ACKNOWLEDGEMENT

The author would like to thank Golnaz Hashemian for the proof reading of the manuscript.

REFERENCES

- [1] A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE*, Vol. 40, pp. 1098-1101, Sept. 1952.
- [2] R. Hunter and A. H. Robinson, "International Digital Facsimile Standard," *Proc. IEEE*, vol. 68, no. 7, pp. 854-867, 1980.
- [3] R. Hashemian, "Memory Efficient and High Speed Search Huffman Coding", *IEEE Transactions on Communications*, Vol. 43, No. 10, pp. 2576-2581, October, 1995.
- [4] R. Hashemian, "Design and Hardware Implementation of a Memory Efficient Huffman Decoding," *IEEE Trans. on Consumer Electron*, vol. 40, No. 3, pp 345-351, August. 1994.
- [5] M. K. Rudberg and L. Wanhammar, "Implementation of a Fast MPEG-2 Compliant Huffman Decoder", *Proc. EUSIPCO '96*, Trieste, Italy, September 1996.
- [6] S. F. Chang and D. G. Messerschmitt, "Designing High-Throughput VLC Decoder Part I Concurrent VLSI Architectures", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 2, No. 2, pp. 187-196, June 1992.
- [7] H. D. Lin and D. G. Messerschmitt, "Designing High-Throughput VLC Decoder Part 11 - Parallel Decoding Methods", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 2, No. 2, pp. 197-206, June 1992,
- [8] S. Ho and P. Law, "Efficient Hardware Decoding Method for Modified Huffman code", *Electronics Letters*, Vol. 27, No 10, pp. 855-856, May 1991.
- [9] A. IC Jain, "Image data compression: A review," *Proc. IEEE*, vol. 69, pp. 349-389, Mar, 1981.
- [10] T. J. Fexguson and J. H. Rabinowitz, "Self-synchronizing Huffman codes," *IEEE Trans. InformL Theory*, vol. IT-30, pp. 687-693, July 1984.
- [11] S. M. Lei and M. T. Sun, "An entropy coding system for digital HDTV applications," *IEEE Trans. Circuit Syst. Video Technol.*, vol. 1, pp. 147-155, Mar. 1991.
- [12] S. M. Lzi, M. T Sun, and & H. Tzou, "Design and hardware architecture of high-order conditional entropy coding for image," *IEEE Trans, Circuit Syst. Video Technol.*, vol, 2, pp. 176-186, June 1992.
- [13] S. Roman, *Coding and Information Theory*. New York: SpringerVerlag, 1992.