Efficient Representation and Decoding of Static Huffman Code Tables in a Very Low Bit Rate Environment

Nick B. Body and Donald G. Bailey
Institute of Information Sciences and Technology, Massey University,
Palmerston North, New Zealand.
E-mail: Nicholas.Body.1@uni.massey.ac.nz, D.G.Bailey@massey.ac.nz

Abstract

The lossless entropy coding used in many image coding schemes often is overlooked as most research is based around the lossy stages of image compression. This paper examines the relative merits of using static Huffman coding with a compact optimal table verses more sophisticated adaptive arithmetic methods. For very low bit rate image compression, the computationally simple Huffman method is shown to be competitive and often superior to adaptive algorithms. We present a method of efficiently representing an optimal Huffman table using delta coded symbol bit lengths. The decoding of the bitstream can also be accelerated by using table lookup operations.

1. Introduction

This paper offers two improvements to the efficiency of use of static Huffman coding in a very low bit rate image codec (coder-decoder) environment. This environment requires the bit rate of the codec to be minimised. The implementation speed also needs to be high to minimise any delay in a system.

Most modern image compression methods use a lossy coder to give good compression, followed by some form of lossless entropy coding performed on the resultant symbol streams to minimise the total output file size [1]. The lossless coding stage typically compresses the symbol stream by about 2:1. It is this lossless coding step that this paper concentrates on, rather than the lossy compression step, although we do make use of our knowledge of the form of the symbol stream [2].

In a very low bit rate environment, there are **three** key areas to be considered when choosing an entropy coder. The coder must closely match the real entropy of the data stream, it must have an efficient implementation to minimise coding delay and the coder must minimise the amount of overhead information to be transmitted to the decoder. Various schemes to implement entropy coding make trade-offs between these key areas.

- Fixed Huffman tables are not optimal for modelling the true entropy of a particular instance of a data stream. Therefore, although there is no overhead required in sending a fixed table, the use of such tables is sub-optimal, and limits the compression obtained.
- Sending an optimal Huffman table creates a significant overhead in a very low bit rate environment.
- By contrast arithmetic coding more closely models the entropy of a data stream. Unfortunately the implementation speed of the bitstream encoding operation is reported to be significantly slower than Huffman coding and the symbol statistics still need to be sent as overhead [3].
- Both adaptive Huffman and adaptive arithmetic algorithms have the advantage of not requiring the overhead of sending a code table. Neither algorithm is as optimal in compression as static coding where the statistics of the symbol stream are stationary. This is because it takes several hundred symbols to be coded before the entropy model converges to that of the static method [3]. In a very low bit rate environment this is particularly significant where there may only be a few thousand symbols to be encoded per image. While adaptive arithmetic coding is more compact than adaptive Huffman coding, the computational overheads are significantly higher because of the need to maintain a cumulative histogram to perform the adaption.

These trade-offs led to the use of a static (non-adaptive) Huffman coding in this application. This ensures that a near optimal model of the entropy of each compressed image is used and the implementation is computationally efficient. The issues are then how to compactly represent the Huffman code table to minimise overheads, and how to efficiently decode the resultant coded symbol stream.

2. Representation of the Huffman table

It is well known that a Huffman table can be uniquely represented by a list of symbol value and frequency count pairs [3]. In many applications the symbol values may

0-8186-8821-1/98 $10.00 Copyright 1998 IEEE

range from 0 to 255 and it may be more efficient to just send the frequency count values for all 256 possible symbols in sequence rather than using a list of pairs. This is efficient when most of the possible symbols are used in the data stream to be coded. If however say only a third of the symbols are used and they are clumped together then a ranged frequency count method can be advantageous [3]. For each range of symbols in the distribution, the ranged method uses two 8 bit numbers to define the end points of the range. This is then followed by the symbol frequency counts within each of the ranges. Such a clumped distribution is common when coding quantised transformed image data that has been run length coded. As can be seen from table 1 **a)** and **b)**, the overhead associated with this approach can be significant with low bit rate coding.

Rather than send the symbol frequencies, only the symbol bit-lengths need to be transmitted to be able to reconstruct the table. The known order of the symbols and their bit lengths allows both the coder and decoder to assign the same bit pattern to the symbols, as described later. Since the symbol lengths are related to the logarithm of the reciprocal symbol frequency, fewer bits are required to represent the lengths. Highly compressed images generally do not require more than 16 bits for the longest Huffman code symbol. It can be proved for symbol streams with less than 6764 symbols, that no symbol will have a bit length of greater than 16 bits [3]. In practice, symbol streams of up to 100,000 symbols did not require symbol codes exceeding 16 bits.

A simple code for the bit lengths is a 5/1 code:

Zero bits0
1 to 16 bits 1nnnn

Since the zero bit symbols are clumped, we now can gain further compression by run length coding the zero bit symbols. This gives a simple 5/range code:

Zero bits0nnnnnnn
1 to 16 bits 1nnnn

These approaches approximately halve the table overhead compared to transmitting the frequency distribution (table 1 **c)** and **d)**).

Finally, the frequency distribution of symbol bit lengths is not uniform as it is related to the branching structure used to represent the Huffman probability tree. This allows the table to be compressed further by using an entropy coding stage on the table. The difficulty here is that the symbol length distribution is quite data dependent, preventing a fixed table from being used, and the overhead of providing the optimal table is significant. However, because the original frequency distribution varies reasonably smoothly, codes for similar symbols have similar lengths. This allows the image dependence to be removed by delta coding the non-zero bit lengths. Tests revealed that the distribution of the delta coded bit lengths does not vary significantly between images allowing a fixed table to be used for these. In over 100 symbol streams tested, all of the deltas were between -5 and +5. However, to be safe, an escape code was provided to explicitly specify the symbol length in the rare occurrence that this range is exceeded. A code was reserved for isolated unused symbols (those with 0 bits), and two codes for identifying short runs (2 to 9) and long runs (10 or more) of unused symbols since short runs appeared relatively frequently. This resulted in another factor of 2 reduction in the overhead of sending the optimum table (see table 1 **e)**). Table 2 shows the fixed Huffman table used to transmit the delta coded symbol length table.

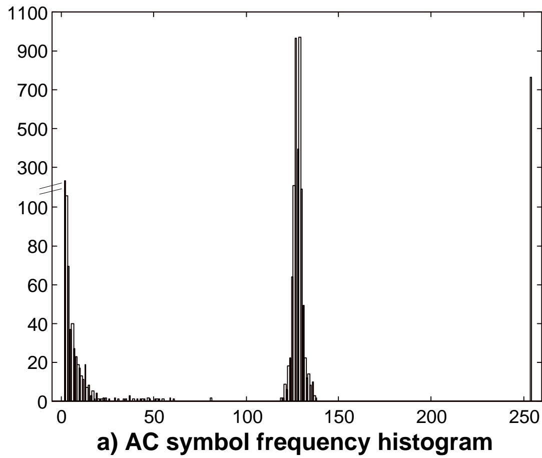
3. Coding results

The relatively low overhead associated with transmitting the optimum Huffman table allowed us to make use of the knowledge of the image statistics to achieve further lossless compression. The symbol stream from a typical wavelet codec consisted of 2 components - an AC quad-tree section, and the DC approximation image. These had quite different characteristics: the DC section was broader and did not contain run length codes. We were able to make a further 3 to 5 percent net gain by coding the AC and DC components independently, and transmitting the optimum Huffman tables for each section.

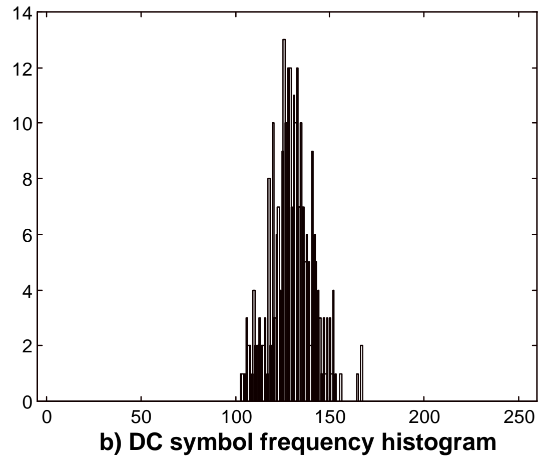
Table 1. Comparison of overheads associated with various Huffman table coding strategies

	Single stream			Splitting stream into AC & DC components				
Huffman table encoding method	**Table size (bits)**	**Total file size (bits)**	**Overhead as % of file size**	**AC table size (bits)**	**DC table size (bits)**	**Total table size (bits)**	**Total file size (bits)**	**Overhead as % of file size**
a) 10 bit AC, 4 bit DC	2560	20461	**13.0 %**	2560	1024	3584	20672	**17.0 %**
b) 10 bit AC, 4 bit DC ranged	1256	19157	**6.6 %**	884	276	1160	18248	**6.4 %**
c) 5/1 coding	648	18549	**3.5 %**	512	472	984	18072	**5.5 %**
d) 5/range coding	620	18521	**3.3 %**	429	307	736	17824	**4.1 %**
***e)** Delta coding*	*384*	*18285*	***2.1 %***	*262*	*179*	*441*	*17529*	***2.5 %***

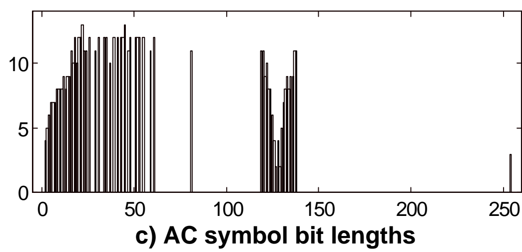
0-8186-8821-1/98 $10.00 Copyright 1998 IEEE



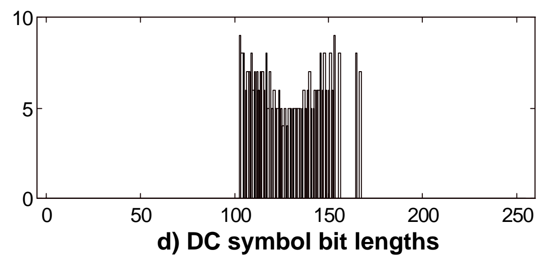
a) AC symbol frequency histogram



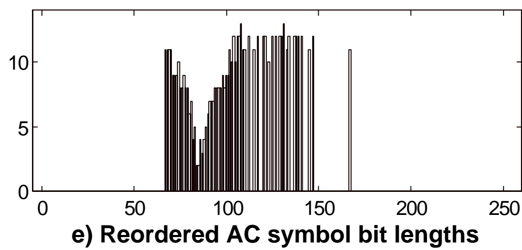
b) DC symbol frequency histogram



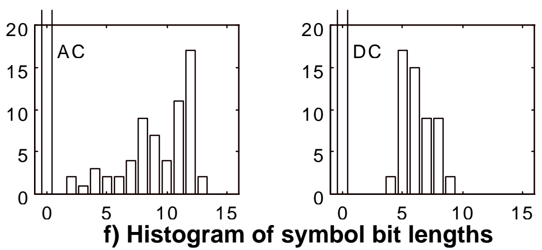
c) AC symbol bit lengths



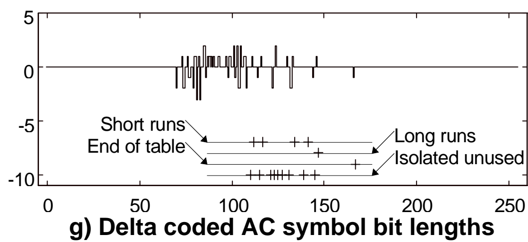
d) DC symbol bit lengths



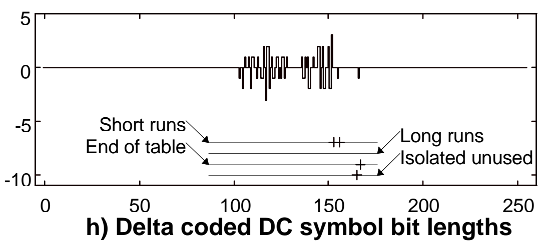
e) Reordered AC symbol bit lengths



f) Histogram of symbol bit lengths

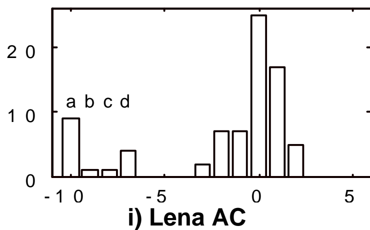


g) Delta coded AC symbol bit lengths

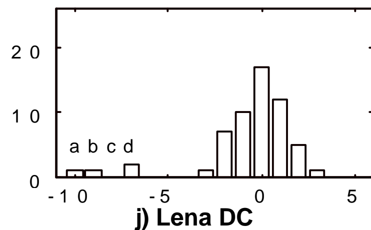


h) Delta coded DC symbol bit lengths

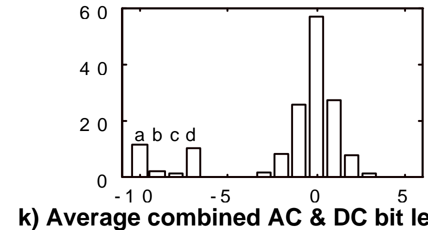
i-k) Histograms of delta coded symbol bit lengths



i) Lena AC



j) Lena DC



k) Average combined AC & DC bit lengths from 100 compressed symbol streams

Escape Symbols: a = Isolated unused, b = End of table, c = Long runs, d = Short runs

Figure 1. Statistics from Lena image at Quantiser 4 setting, 256^2 pixels, 4462 AC symbols & 256 DC symbols

0-8186-8821-1/98 $10.00 Copyright 1998 IEEE

Table 2. Delta table code allocation

Symbol	Bits	Code
Isolated unused	4	1100
End of table	7	1111100
Long runs	8+7	11111110nnnnnnn
Short runs	4+3	1101nnn
Explicit length	12+5	111111111111nnnnn
Delta -5	12	111111111110
Delta -4	9	111111110
Delta -3	7	1111110
Delta -2	4	1110
Delta -1	3	101
Delta 0	1	0
Delta +1	3	100
Delta +2	5	11110
Delta +3	7	1111101
Delta +4	10	1111111110
Delta +5	11	11111111110

Figure 1 shows the process of compressing the optimum Huffman tables for a 256^2 pixel Lena image wavelet coded to give 4462 AC and 256 DC symbols. The compressed image is shown in figure 5. The symbol frequency histograms, **a)** and **b)**, show that the two distributions are significantly different. These distributions are Huffman coded to give the lengths of each symbol, **c)** and **d)**. Since the AC section contained both symbols and run-length symbols, the symbols were reordered, **e)**, to give a tighter delta distribution. The histograms of the symbol bit lengths, **f)**, show that the distributions are significantly different between the AC and DC components. There is also a similar variation between images and between the output from different quantisers when the same image is used. The bit lengths are then delta coded, **g)** and **h)**, to give similar looking tight distributions, **i)** and **j)**. The fixed table for coding the delta coded symbol lengths (table 2) was derived from the statistics of 100 symbol streams. These were made of from 10 different images each at 3 different resolutions (512^2, 256^2, and 160^2) using a range of quantisers. The average delta distribution is shown in figure 1 **k)**.

To verify our claims that adaptive methods are not as suitable as using the optimal table in a very low bit rate environment, we compared the resultant file sizes for 100 symbol streams. As expected, the adaptive arithmetic method always outperformed the adaptive Huffman method. On average, the optimum Huffman method and the adaptive arithmetic method performed similarly, although our method tended to do better for smaller symbol streams (less than about 8000 symbols) and the adaptive arithmetic method was better for larger symbol streams (figure 2). A detailed breakdown of the overheads is shown in figure 3 for the Lena image. The overheads for the adaptive methods are the effective overheads obtained by comparing the file size if the

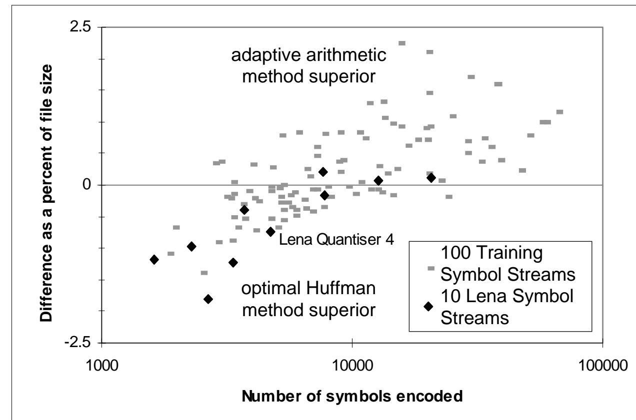


Figure 2. Difference in file sizes between optimal Huffman and adaptive arithmetic coding as a function of the number of symbols encoded

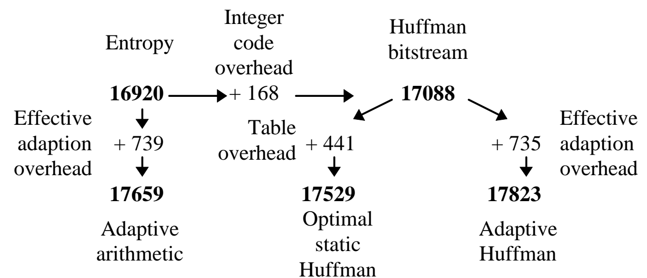


Figure 3. Detail of overheads involved in entropy coding the Lena Quantiser 4 symbol stream, all numbers are in bits

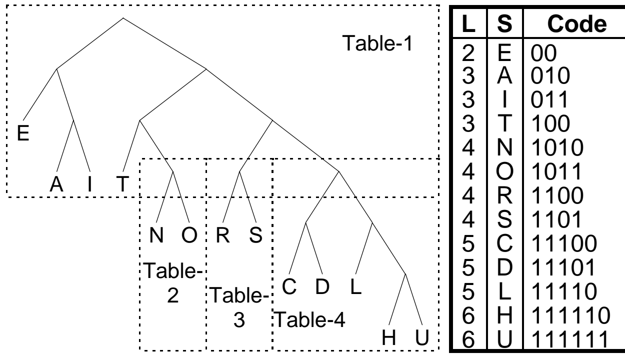
distribution was known exactly with that obtained by using adaption. The overhead results from the coder building up a model of the symbol stream.

4. Assigning bit patterns from symbol lengths

Our method relies on being able to generate a unique code for each symbol based solely on knowing the symbol length. The procedure is as follows:

1) Sort all the symbols (**S**) in ascending order of symbol length (**L**)
2) Where several symbols have the same length, these are sorted in symbol order
3) Starting with top of the list (the shortest symbol) assign the code of all zeros
4) For each symbol down the list, assign the next available code in increasing sequence

0-8186-8821-1/98 $10.00 Copyright 1998 IEEE



L	S	Code
2	E	00
3	A	010
3	I	011
3	T	100
4	N	1010
4	O	1011
4	R	1100
4	S	1101
5	C	11100
5	D	11101
5	L	11110
6	H	111110
6	U	111111

Figure 4. a) Huffman tree with table decoding partitions b) Code allocation

Figures 4 **a)** and **b)** show an example of the method applied to a small set of symbols. This method assigns a unique code to each symbol, and also has the advantage that the longer codes are at the end of the table. This makes the decoding more efficient.

5. Decoding

A computationally efficient method for decoding Huffman bitstreams is proposed. Normally each bit received is used to guide the traversal of a decoding tree similar to that shown in figure 4 **a)**. Our method uses a set of hierarchical lookup tables as shown in table 3. Three bits from the bitstream are used at a time to form an index into the root table (Table-1). For most of the frequently occurring symbols, this initial index will resolve the symbol and the number of bits consumed. For less frequent codes the index will result in a second indexing operation into a child table with the next three bits from the bitstream used as the index. The consumed bits are removed from the bitstream prior to decoding the next symbol.

In practice, rather than using three bits for an index, we use eight bits, giving a 256 element table. This allows up to eight bits to be removed every lookup, rather than one bit per lookup using a conventional tree structure.

Table 3. Table partition for decoding

Index	Table-1	Table-2	Table-3	Table-4
000	E(2)	N(4)	R(4)	C(5)
001	E(2)	N(4)	R(4)	C(5)
010	A(3)	N(4)	R(4)	D(5)
011	I(3)	N(4)	R(4)	D(5)
100	T(3)	O(4)	S(4)	L(5)
101	Table-2	O(4)	S(4)	L(5)
110	Table-3	O(4)	S(4)	H(6)
111	Table-4	O(4)	S(4)	U(6)

6. Conclusions

The simplicity of using static Huffman coding for the lossless stage of image compression at very low bit rates is appealing. This paper reveals that this computationally efficient method when combined with a compact Huffman table representation is competitive when compared with more sophisticated methods such as adaptive arithmetic coding. The implementation speed of Huffman coding and decoding is significantly faster than adaptive and arithmetic methods.

7. Acknowledgements

The authors would like to thank Professor R.M. Hodgson and Dr W.H. Page for their assistance in preparing this paper.

8. References

[1] R.J.Clarke, "Image & Video Coding: Development and Prospects", in Proceedings of the first joint Australia and New Zealand biennial conference, DICTA'97, IVCNZ'97, Albany, Auckland, New Zealand, pp.1-10, 1997. ISBN 0-473-04947-3.

[2] N.B.Body, W.H.Page, J.Y.Khan & R.M.Hodgson, "Efficient Mapping of Image Compression Algorithms on a Modern Digital Signal Processor", in Proceedings of the 4th Annual New Zealand Engineering and Technology Postgraduate Students Conference, University of Waikato, New Zealand, pp.59-64, 1997. ISBN 0-473-04578-8.

[3] M.Nelson & J.Gailly, "The Data Compression Book", 2nd ed. M&T Books, pp.19-136, 1996.

Figure 5. Lena image at Quantiser 4 setting, 256^2 pixels, compressed by 30:1

0-8186-8821-1/98 $10.00 Copyright 1998 IEEE