Huffman code efficiencies for extensions of sources.

Peter Fenwick

Technical Report 89

ISSN 1173-3500

Original version prepared February 1994

Department of Computer Science, The University of Auckland,
Private Bag 92019, Auckland, New Zealand

p.fenwick@auckland.ac.nz

Abstract

It is well known that the efficiency of noiseless coding may be improved by coding extensions of the source; for large extensions the efficiency is arbitrarily close to unity. This paper shows that the efficiency is not always improved just by coding the next extension. In some cases the code of a larger extension is markedly less efficient than its predecessor, although always within the theoretical limits of efficiency. We show how the phenomenon arises from changes to the Huffman coding tree as the source probabilities change and investigate it for binary and ternary codes.

Introduction.

For a discrete memoryless information source S described by a source alphabet of symbols occurring with probabilities $\{P_1, P_2, P_3, \ldots\}$, the entropy per source symbol is $H(S) = -\sum P_i.\log(P_i)$. When coding symbols from S for transmission over a noiseless channel it is usual to employ a variable length code, choosing shorter codewords for the more probable symbols. Given that each codeword has the probability P_i and length L_i the expected length of the resultant code is $L = \sum P_i L_i$. A fundamental result of information theory is that $H(S) \leq L$. Alternatively, if we define the code efficiency to be $\eta = H(S) / L$, then $\eta \leq 1$. A second important result, known as Shannon's first theorem, or the "noiseless coding theorem" (Abramson[1], p72), is that we can improve the coding efficiency by coding extensions of the source, i.e. grouping source symbols in groups of 2, 3 or more symbols and encoding the composite symbols. For the nth extension (coding n source symbols at a time) the efficiency is bounded by $1 \geq \eta > 1 - 1/n$. Thus by encoding a sufficiently large extension, the efficiency can be forced arbitrarily close to unity.

The usual code in this situation is the Huffman code[4]. Given that the source entropy is H and the average codeword length is L, we can characterise the quality of a code by either its efficiency ($\eta = H/L$ as above) or by its redundancy, $R = L - H$. Clearly, we have $\eta = H/(H+R)$. Gallager [3] shows that the upper bound on the redundancy is P + 0.0861, where P is the probability of the most frequent symbol. Johnsen [5] and Capocelli etal [2] derive progressively tighter bounds to the redundancy, developing relations for different ranges of P.

The above authors study only the "base" Huffman code, although a specified extension can be treated much as a simple code in its own right. What does not seem to have been examined as extensively is the behaviour as we code successively higher extensions of a given source alphabet. A naive application of the above bound ($1 \geq \eta > 1 - 1/n$) shows that the efficiency should always improve as higher-order extensions are coded. The efficiency is generally assumed to approach the ideal quite quickly as source extensions are encoded (Abramson[1], p 87). Thus, if we consider the binary source P = {0.85, 0.15}, encoded with a binary Huffman code, we find that the efficiencies of the first few extensions are as shown in Table 1. The approach to the ideal is obviously quite rapid.

Extension	Efficiency
1	0.6098
2	0.8544
3	0.9663
4	0.9837
5	0.9926

Table 1. Efficiencies for extensions of the source {0.85, 0.15}

Some Irregular Cases

Although Shannon's Theorem places bounds on the efficiency of a reasonable compact code, it says very little about the behaviour of an actual code. With slight changes to the source probabilities of Table 1, the Huffman code efficiencies for the binary source P = {0.8, 0.2} are shown in Table 2. For the first three extensions the efficiency improves as we would expect, but at the fourth extension it deteriorates markedly. The fifth extension is little better than the fourth and both are markedly worse than the the third. Even so, the code efficiency is still well within the theoretical bounds. Cases such as this, where the efficiency decreases with increasing extension, will be termed "irregular" and the contexts in which they occur "irregularities". An "irregular extension" is one which gives a poorer efficiency than its immediate predecessor.

Extension	Efficiency
1	0.7219
2	0.9255
3	0.9917
4	0.9745
5	0.9783

Table 2. Efficiencies for extensions of the source {0.80, 0.20}

To find other irregularities, a search was made of Huffman codes (binary codes of binary sources) for probabilities {ω, 1–ω} with ω varying from 0.05 to 0.50 in steps of 0.05 and for all extensions up to the 7th. The results are summarised in Table 3, with the efficiencies underlined for the irregular extensions. The irregularities all occur over ranges of extensions and probabilities, but all areas are included in the table. We also see that even the example given first is poorly-behaved as soon as we look beyond the range shown in Table 1! In general we see that if an extension results in a particularly good code, it may be counter-productive to attempt to use a higher extension.

ω	Ext = 1	2	3	4	5	6	7
0.05	0.2864	0.4992	0.6610	0.7800	0.8519	0.9056	0.9429
0.10	0.4690	0.7271	0.8805	0.9522	0.9767	0.9975	0.9887
0.15	0.6098	0.8544	0.9663	0.9837	0.9926	0.9817	0.9914
0.20	0.7219	0.9255	0.9917	0.9745	0.9783	0.9954	0.9866
0.25	0.8113	0.9615	0.9859	0.9913	0.9921	0.9910	0.9932
0.30	0.8813	0.9738	0.9699	0.9882	0.9913	0.9922	0.9964
0.35	0.9341	0.9692	0.9840	0.9836	0.9963	0.9896	0.9963
0.40	0.9710	0.9710	0.9894	0.9896	0.9931	0.9945	0.9955
0.45	0.9928	0.9928	0.9928	0.9929	0.9946	0.9951	0.9959

Table 3. Efficiencies for extensions of a range of binary codes

Analysis of the behaviour

The previous paragraph shows that the efficiency sometimes decreases with higher extensions. The reason for this behaviour lies in the generation of the Huffman code and the variation of that coding as the symbol probabilities vary. The generated Huffman code, the shape of its associated tree and the distribution of codeword lengths are all critically dependent on the actual symbol probabilities. Each tree is optimum at one set of probabilities; as we move away from those values the coding will deteriorate. In many cases the coding with another tree will be improving until, when the two are equal, the code will "flip" to the other tree and its pattern of codeword lengths. We will thus get a discontinuity in the graph of efficiency against symbol probability as one tree takes over from the other.

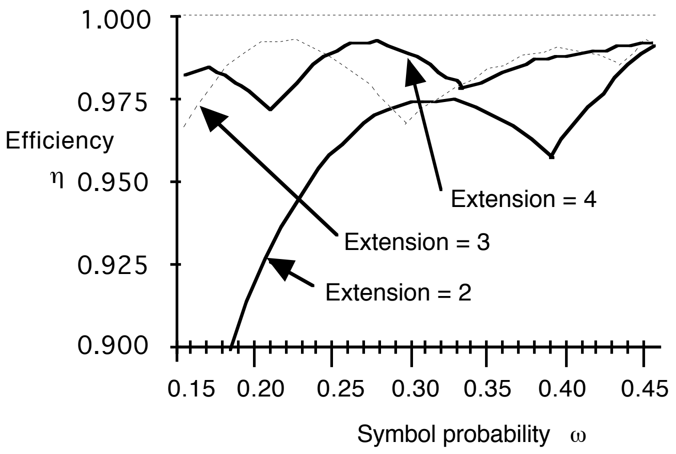

Figure 1.
Huffman code efficiency; extension as parameter

Figure 1 shows the efficiencies of binary Huffman codes for extensions up to the fourth and for a range of symbol probabilities. It is clear that each curve is a combination of several convex functions, corresponding to the different coding trees as discussed above. The simplest case is the second extension, the lowest line in Figure 1, which has a discontinuity at about ω = 0.4. By examining actual codes, we find that for smaller values of ω the codewords have lengths of {3, 3, 2, 1}, while for larger values all codewords have a length of 2. The codes for symbol probabilities of

{0.35, 0.65} and for {0.45, 0.55} are shown in Table 4; the table also includes the formulae for the average codeword length of the first code as a function of ω. Combining the formulae and simplifying gives the average codeword length as $L = -\omega^2 + 3\omega + 1$. The two codings give identical average lengths when L = 2, the transition between the two alternatives occurring at ω = 0.38196. Similar, but more complex, analyses apply to the higher extensions, with that for the third extension given later.

	P(A) = 0.35; P(B) = 0.65		Length for P(A) = ω	P(A) = 0.45; P(B) = 0.55	
BB	0.4225	0	$(1-\omega)^2$	0.3025	11
BA	0.2275	10	$2 \times (1-\omega)\,\omega$	0.2475	10
AB	0.2275	111	$3 \times \omega\,(1-\omega)$	0.2475	01
AA	0.1225	110	$3 \times \omega^2$	0.2025	00
			Average Length = $-\omega^2 + 3\omega + 1$		

Table 4.Huffman codes for probabilities of 0.35 and 0.45

More insight may be gained by considering Figure 2, which shows various functions of the second extension of a binary Huffman code.

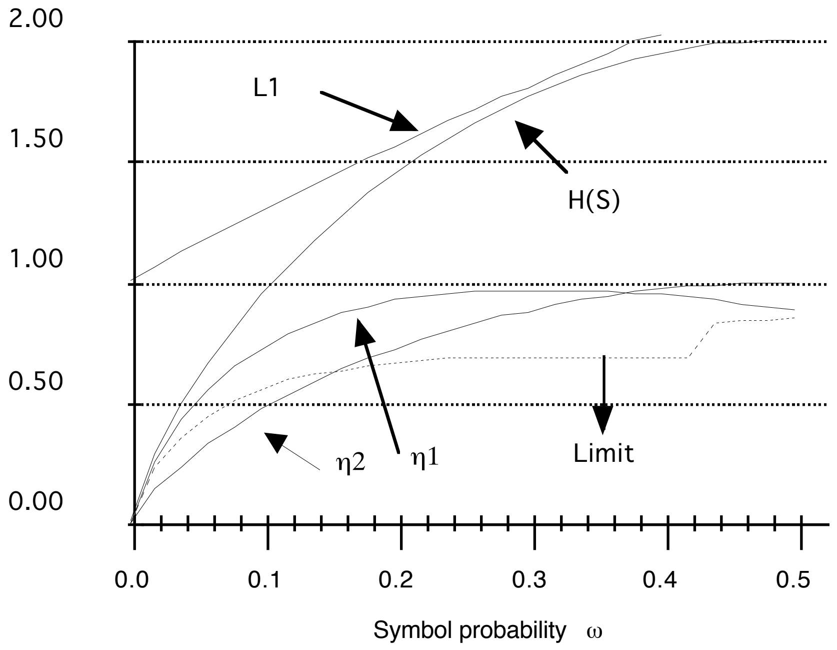

Figure 2. Details of efficiency , second extension of binary code

The values which are shown are –

H(S)	the entropy of the second extension of the source
L1	the average length of the code with word lengths {3, 3, 2, 1}. (This is actually an inverted parabola, but the short segment shown here appears as almost a straight line.)
η1	the efficiency of the code using word lengths {3, 3, 2, 1}. (η1 = H(S)/L1)
η2	the efficiency of the code with all words of length 2.
Limit	the lower bound to the efficiency, using the result of Capocelli etal.

The efficiency of the final code is simply the envelope of the efficiencies for the two codeword configurations. Except for probabilities close to zero, it is always much better than the bound of Capocelli etal. The theoretical bounds still apply, but the variations within those bounds lead to the "unexpected" behaviour.

Returning to Figure 1, we see that the third extension is characterised by several patterns of codewords. The details of these patterns are given in Table 5. (We start by giving approximate ranges only; the transition between the second and third cases is barely visible in the diagram.) For each range we give the formula for the average length as a function of ω. As with the second extension, transitions from one coding to the next occur when the corresponding lengths are equal, the actual values being ω = 0.2929, ω = 0.3333 and ω = 0.4302.

```
                              P(A)
            0.1       0.2       0.3       0.4       0.5
P(B)----:----|----:----|----:----|----:----|----:----|
0.01 ·····························44··············4444
0.02  ··························44············44444444
0.03   ·····················44444············44444444
0.04    ··················4444444···········444444444
0.05     ···············4444444444··········44444444
0.06      ············4444444444444·······4444444444
0.07       ··········444444444444444······444444444
0.08        ·········44444444444444············44··
0.09         ········4444444444444················
0.10          ·······444444444444·················
0.11           ······4444444444··················
0.12            ······44444444···················
0.13             ······44444··33················
0.14              ······44·333333···············
0.15               ······33333333··············
0.16                ···3333333333··············
0.17                 333333333333344··········
0.18                  3333333333334···········
0.19                   33333333333···········
0.20                    3333333333···········
0.21                     333333333··········
0.22                      3333333···········
0.23                       33333···········
0.24                        33·············
0.25                         ·············
0.26                          ············
0.27                           ··········
0.28                            ·········
0.29                             ·······
0.30                              ······
0.31                               ····
0.32                                ···
0.33                                 ·
```

Figure 3. Irregularities for ternary Huffman codes

Ternary Codes

Having investigated the behaviour of binary Huffman codes, we now extend the analysis to ternary (base 3) codes, with the results shown in Figure 3 for the source S={A, B, C}. To simplify the presentation we display only the regions where the performance deteriorates, showing the number of the irregular extension and ignoring the magnitude of the deterioration. The range of probabilities covers all combinations, given that the third probability is implied and that the three probabilities are interchangeable.

The digits in Table 6 show the irregular extensions for each combination of probabilities. The greatest changes are those shown in underlined bold-face. In going from the 2nd to the 3rd extension, the greatest deterioration occurs at {0.20, 0.20, 0.60} where the efficiency drops from 0.9829 to 0.9512, a change of 0.0317. The corresponding change for the 3rd to 4th extension occurs at {0.08, 0.23, 0.69}, a fall from 0.9985 to 0.9597, or change of 0.0388.

Conclusions

We have shown that coding a higher extension of a Huffman code does not necessarily improve the code efficiency. In all cases the coding with a particular Huffman tree is optimal for only a range of source probabilities and deteriorates as the source probabilities deviate more from the optimum. Eventually another tree will give better performance and the coding will change to use the new tree. The code efficiency may be relatively poor in the region of the transition. If the probability at the transition is close to an optimum probability for the previous extension, the performance may deteriorate when moving to the higher extension. In all cases the efficiency is well within the known theoretical bounds.

Final note

All copies of the original of this report appear to have been lost. This document represents a good approximation of that original report. A salvaged version was prepared September 26, 1997, and then reformatted on October 31 2007.)

A version appeared in IEEE Trans Comm, Vol 43, No 2/3/4, Feb/Mar/Apr 1995 pp 163–165 (but unfortunately with diagrams confused).

References

[1] N. Abramson, "Information Theory and Coding", McGraw-Hill, 1963

[2] R.M. Capocelli, R. Giancarlo, I.J. Taneja, "Bounds on the redundancy of Huffman codes", *IEEE Trans. Inform.. Theory,* Vol. IT-32, no 6, pp 854-857, Nov. 1986.

[3] R.G. Gallager, "Variations on a theme by Huffman", *IEEE Trans. Inform.. Theory,* Vol. IT-24, no 6, pp 668-674, Nov. 1978

[4] D.A. Huffman, "A method for the construction of minimum redundancy codes", *Proc. IRE*, Vol. 40, pp 1098-1101, 1952

[5] O. Johnsen, "On the redundancy of binary Huffman codes", *IEEE Trans. Inform.. Theory,* Vol. IT-26, no 2, pp 220-223, Mar 1980