Anomalous Efficiencies of Extended Huffman Codes

Abstract

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

Keywords Huffman codes, extended codes, efficiency, anomalies

Author Peter M. Fenwick, Member IEEE

Affiliation Department of Computer Science
University of Auckland
Auckland, New Zealand.

Postal Address Dr P.M. Fenwick
Dept of Computer Science
University of Auckland
Private Bag 92019
Auckland
New Zealand.

E-mail `p_fenwick@cs.auckland.ac.nz`

Telephone + 64 9 373 7599 ext 8298

FAX + 64 9 373 7453

Anomalous Efficiencies of Extended Huffman Codes

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

Introduction.

For an Information Source S described by a Source Alphabet of symbols occurring with probabilities $\{P_1, P_2, P_3, \ldots\}$, the average information per source symbol is defined to be the Source Entropy $H(S) = -\sum P_i.\log(P_i)$. When coding symbols from S for transmission over a noiseless channel it is usual to encode each source symbol (with probability P_i) into a codeword with length L_i, choosing shorter codewords for the more probable symbols. Given that each codeword has the probability P_i the average length of the resultant code is $L = \sum P_i L_i$. A fundamental result of Information Theory is that $H(S) \le L$. Alternatively, if we define the code efficiency to be $\eta = H(S) / L$, then $\eta \le 1$. A second important result (Shannon's First Theorem) is that we can improve the coding efficiency by coding extensions of the source, ie 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 $1 \ge \eta > 1- 1/n$. 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[3] and its performance has been investigated by several authors. 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 [2] shows that the upper bound on the redundancy is $P + 0.0861$, where P is the probability of the most frequent symbol. Johnsen [4] and Capocelli etal [1] 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. Simple theory says that the efficiency will improve as higher-order extensions are coded and it is generally assumed to approach the ideal quite quickly as source extensions are encoded. 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.

Some Anomalous 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. Note though that, even with these anomalies, the code efficiency is still well within the theoretical bounds.

To determine whether this result is an isolated one, a search was made of Huffman codes (binary codes of binary sources) for probabilities {□, 1–□} with □ 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 all cases where the efficiency is less than for the previous extension. (Only cases where the decrease is greater than 0.0001 are indicated.) The anomalies 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.

An alternative presentation in greater detail is in Table 4, with □ varying from 0.01 to 0.5 in steps of 0.05, extensions up to 9 and showing the amount by which a code is worse than for the immediately preceding extension with that probability. The digit which shows the error is calculated by the formula $15 + \log_2(v)$ where v is the change in efficiency relative the previous extension; it is approximately the number of low-order bits which differ in the 4th decimal digit. (The scaling is actually chosen to allow the anomalies to be displayed as single digits.)

Analysis of the behaviour

The answer lies in the coding 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 is 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 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 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-upward functions, corresponding to the different coding trees as discussed above. The simplest case is the second extension, the dotted line in Figure 1, which has a discontinuity at about □ = 0.4. By examining actual codes, we find that for smaller values of □ 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 5; the table also includes the formulae for the average codeword length of the first code as a function of □. Combining the formulae and simplifying gives the average codeword length as L□□□^{2} + 3□ + 1. Setting L = 2 gives the condition that the two codings give identical average lengths and locates the discontinuity where the coding switches between the two alternatives. Solving the quadratic equation shows that the transition occurs at □ = 0.38196. Similar, but more complex, analyses apply to the higher extensions; that for the third extension is given later.

More insight may be gained by considering Figure 2, which shows various functions of the second extension of a binary Huffman 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.)
□1	the efficiency of the code using word lengths {3, 3, 2, 1}. (□1 = H(S)/L1)
□2	the efficiency of the code with word 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 there is some degree of chaotic behaviour within those bounds.

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 6. (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 □. As for the second extension, transitions from one coding to the next occur when the corresponding lengths are equal. The transitions occur at □ = 0.2929, □ = 0.3333 and □ = 0.4302.

Ternary Codes

Having investigated the behaviour of binary Huffman codes, we now extend the analysis to ternary (base 3) codes. One problem is that high extensions of non-binary codes very soon include an enormous set of codewords, over 2000 for the 7th extension of a ternary code, making the codes expensive to generate. Another major problem lies in the presentation of the results as there are now three probabilities and the extension, to give three independent variables. To simplify the presentation we display only the regions where the performance deteriorates, showing the number of the anomalous extension and ignoring the magnitude of the deterioration.

Considerable simplification results from the symmetries of the probabilities. Given that the source alphabet S = {A, B, C}, we have that P(A) + P(B) + P(C) = 1.0, and can immediately write P(C) = 1.0 – P(A) – P(B). There are therefore only two independent probability variables and they can be represented as the axes of a conventional graph. Because all of the probabilities are interchangeable we can take advantage of further symmetries; for the first probability it is sufficient to cover the range $0 < P(A) < 0.5$ and for the second $0 < P(B) < P(A)$, up to maximum value of 0.25.

The results are given in Table 7, where the digits show which extension gives poorer performance than its predecessor for a 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.

The regions of anomalous behaviour have been investigated for binary codes (up to the 9th extension) and for ternary codes.

References

[1] 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.

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

[3] D.A. Huffman, "A method for the construction of minimum redundancy codes",

Proc. IRE, Vol. 40, pp 1098-1101, 1952

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

Figure captions

Table 1. Efficiencies of {0.85, 0.15} Huffman code extensions

Table 2. Efficiencies of {0.80, 0.20} Huffman code extensions

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

Table 4. Efficiency losses for extensions of a range of binary codes

Table 5. Huffman codes for probabilities of 0.35 and 0.45

Table 6. Code trees and word lengths for third extension

Table 7. Anomalies for ternary Huffman codes

Figure 1. Huffman code efficiency; extension as parameter

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