Improving Huffman coding

David Wheeler

March 1997

Introduction

It is often stated that Huffman coding is perfect, without noting the strong assumptions :
The coding is character by character with no "state".
The cost of defining the code is neglected.

If these assumptions are broken we can usually improve the coding. If we allow state then arithmetic coding can be used with a potential wastage of one bit per message, rather than one bit per character.

It is to be noted that the cost of definition be-it adaptive or direct is more for arithmetic coding than Huffman. Also, that Huffman coding is usually much faster than arithmetic coding which uses multiplication and division and usually searching as well.

One weakness of the Huffman technique occurs when a probability is in excess of 1/2. There are many methods for dealing with this. Typically, we group the high probability elements, causing the group frequencies to be much less than one half. This essentially introduces a little state and larger definition tables.

Run length coding

A simple way of dealing with this is run length coding, where the length of the run is coded. If the probability is close to $2^{-1/2^n}$, we can code as follows :

a 0 reresents 2^n zeros (assuming 0 most frequent).
a 1 followed by n bits representing m a number less than 2^n
signals m zeros follow by a 1.

The coding can often be improved by Huffman coding the $2^n + 1$ codes using their frequency distribution.

1/2 method

This method covers a greater range than run length and also utilises correlations of successive values.

We use one extra symbol which indicates a pair of the most frequent symbol. This would work quite well over a limited range. The range is greatly extended by using the two symbols which represent one or two of the most frequent character to express in binary a number m. The length of the run of the character is given by extending m by a 1 at the more significant end and subtracting 1.

This is adaptive over a large range of probabilities and uses to some extent the correlation if most frequent character is more likely to occur after such a character.

The method is easily extended to work in any base. For example, we can use the character and three extra symbols for groups of 2,3,4. The sequence can represent a number from which we can work out the size of the total group. If the probability is very high, then a higher base is advantageous.

Use of multiple tables.

If a probability is near $2^{n-1/2}$ then it will be badly coded in a stateless coding regime, leading to a loss of about 1/12n bits. We can use multiple tables and a little state to reduce the loss.

Example :
Assume we have two characters whose probabilities are about $2^{n-1/2}$. We normally would code one with n bits and one with n+1 bits. However, if we use small groups, the frequencies of these two will fluctuate. We therefore look ahead at the next small group and choose the most frequent to have the shorter code. Thus we emit an extra bit every so often to indicate which coding to use.

There is a best length for the groups, probably 8-16. The gain per occurrence decreases like $1/\sqrt{m}$ and it must exceed the one bit cost of selection. This will give a gain on the average. This was assuming the group size was preselected.

An alternate method is to continue with the current selection until the gain over the Huffman case is 1 bit. Then select the code which attains this limit in the shortest string, and continue.

One variant is to have s tables and each time select the best table for the next group. The tables can be found by some iterative process. For example, using one set of tables, select each group and form new frequencies for each group. Stop when no gain, when gain less than say one millibit.

This works quite well for the Block encoding method.

A further variant, is to group perhaps 2m codes of probs. $2^{n-1/2}$. If initially m were coded with n bits and m with n+1 bits, we can select when the shortest string and one selection gains m bits. The selection should be m of length n and m of length n+1, and can be crudely specified by 2m bits. A single coding and decoding table will suffice, with the ability to swop characters.

Of course, we can go further and replace three code of length n by one of length n-1 and two of length n+1. This will lead to a more drastic reconfiguration of the tables, but does cater for the profile of the frequencies changing. It can be catered for by selecting complete tables for coding and decoding.

Cost of the Huffman tables.

The naive approach is to send for each character the length and code. This can usually be improved by having a known alphabet, sending the length for each character code, and selecting a standard code from the many which exist. We can always change the codes for the characters having the same code length.

A useful standard is to have the codes corresponding to the same length in alphabetical order, and to have the smallest code values for the longest codes. This aids the decoder. It can subtract lowest code of each length from the given data, until it finds the length. The offset that remains after the subtraction and that can be used to index a list containing the decodes.

For block compression, after the move to front, the code lengths are almost monotonic and can be coded using that fact. There are some divergences. Usage gaps in the alphabet, break the monotony and also require coding, possibly being given the value of one more than the maximum length. They have a second bad effect in that the first use of the high alphabetic values are encoded by a frequency of average value one. If the gaps were abolished, they would be coded by a value that was probably used many time.

In bred3 a usage table was transmitted, and hopefully its cost was offset by the saving in the first use of some characters, and the lack of need to code the gaps.

Selector Coding.

There is considerable correlation between successive selection codes. We can use a move to front cobined with 1/2 coding to minimise the small cost. It does slightly speed up the decoding process.

Use of Huffman or Arithmetic coding with multiple frequently selected tables

Introduction

The coding efficiency of Huffman or Arithmetic etc. coding can be enhanced by selecting a coding table from a group of tables numbering say gno every gsz characters.

Basics

We assume that the tables have been generated by a search procedure to be fairly good. If the data is random, the distribution frequencies over groups of size gno, also vary randomly. we can sort these group frequencies into groups close together having their best coding and decoding table. The tables let us track the major fluctuations more closely at the cost of a selector every gno characters. The selector can be encoded reasonably, but will have less than $\log_2$ gno bits. For arithmetic coding there will be gain but it will be more than offset by the extra selection bits. For Huffman, there can be an overall gain of no more than the Huffman coding inefficiency. For both cases there will be a gain if the underlying frequencies change as we progress. It is a form of adaptive coding.

Analysis

Random case.
Each group will have a frequency table, and the frequencies will fluctuate from group to group. Given a frequency of f we can expect changes about $df = \sqrt{f}$ leading to losses in coding of $f(df/f)^2 = df^2/f =$ one nit per component loss from the long term average rather than about actual group frequency. By having two tables optimally chosen we can expect to reduce the average loss. For the Huffman case we are guarranteed a gain for large files if the coding was suboptimal. Choose two or three frequencies which are not near powers of 2. and make two tables which exploits a perturbation in which two or three slight variations are made to those frequencies before the code generation is done. Some groups will be encoded better by the perturbed code and the larger gno is the larger the gain which is offset by the selector cost. We can probably choose so that enough are better so we can code the selection and still gain.

An alternate approach

There is a second approach to yield similar results. Instead of having a set of tables, we have a single table and make minor adjustments. The simplest is to interchange the codes of a symbol having n bits with n+1 bits. They could, in a bad case, have almost the same overall frequency. We now use a version of the table until we have gained say four bits on the other version. We now select the best table for the next usage. The result is variable lengths of the different codes for those two symbols, and a forced gain of one bit per usage. As the same table is likely to be used again there will be considerable correlations between successive usage bits, and their coding can be improved.

The selection can be done by the coder so the speed of the decoder will not be reduced very much.

We can simultaneously have many pairs, so their gains aggregate. We can also make more complicated changes while retaining the utility of a single overall permutation. We can take 3 symbols encoded in n bits, and change so that one becomes n-1 bits while the other two become n+1 bits. Thus we have a choice of 4 at each selection point. In this case the profile of the lengths of the codes will have been changed, so the change is more general, but not equivalent to having four complete codes. In some respects if we run r sets of changes it is equivalent to having 2^r tables, and the length of the usages optimised. However, the changes are less general, so I suspect the methods may be practically equivalent.

Results

With block encoding and 1/2 encoding of strings we can gain on the average over the CALGARY files as follows. The first is arithmetic gain over arithmetic coding the second line is the Huffman

Gains in millibits for Arithmetic and Huffman coding

The first of each pair gives arithmetic gain and the second gives the gain respect to Huffman coding. The first pair is the mean for the Calgary file set. The second pair is an 85 character randon set. The third is a 48 character set, the forth is a 128 character set. A crude allowance is made for the cost of the extra coding tables, 210 bits each, and the selectors every bsz characters.

bsz	3000				1000				300				100				50			
bno	5	4	3	2	5	4	3	2	5	4	3	2	5	4	3	2	5	4	3	2
	68	66	66	62	88	85	81	67	99	97	91	74	109	106	97	79	113	107	101	81
	69	69	69	66	90	86	85	70	100	97	93	77	107	106	96	77	108	106	99	78
	-4	-3	-1	0	-3	-2	-1	-1	-3	-4	-3	-1	-9	-8	-6	-5	-17	-17	-13	-9
	16	15	13	7	23	21	19	14	30	31	26	20	39	38	33	29	35	37	36	34
	-6	-3	-1	1	-5	-2	-1	1	-5	-3	-1	2	-8	-6	-2	3	-16	-12	-13	-9
	7	8	6	5	16	16	15	12	24	22	17	13	31	27	18	-8	23	24	26	23
	-6	-4	-2	-1	-5	-3	-2	-1	-4	-3	-2	-1	-6	-5	-5	-3	-12	-12	-10	-6
	-5	-3	0	3	-9	-6	-5	-3	-4	-4	0	-3	-14	-12	-7	-3	-27	-21	-14	-9

It will be noted there is appreciable gain even with two coding tables while increasing the number is more effective with small group sizes. Note that with an optimally coded table of 128 random data the Hnffman coding is perfect and cannot be improved. With 48 or 85 the Huffman is about maximally inefficient losing about 80 millibits. The usual loss is about 30 millibits. The top pair show the gain on the Calgary set due to changing probabilities.