IBM Technical Disclosure Bulletin Vol. 23 No. 2 July 1980

DIGRAMATIC TEXT COMPRESSION

D. A. Hamilton, P. R. Herrold and M. J. Ossefort

Text is normally represented by 8-bit characters. When storage reduction is required, the character set might be limited to just the 26 letters. Then, using 5 bits/character, the 32 states provided are more than enough for the 26 characters.

But the amount of information (measured in bits) that is conveyed by a character is inversely proportional to its probability, rather than just the number of other characters in the alphabet. The number of bits needed to represent a given character is $- \log_2(p)$, where p is the probability of occurrence of the character. For example, a character which occurs half the time would require exactly one bit; if it occurred one-fourth of the time, it would need two bits. The number of bits required to represent a character "on the average" is calculated as the sum of the products of the individual character probabilities and the number of bits required for the given character. (That is, it is a "weighted" average.)

Thus, when still more reduction is needed, Huffman encoding is popularly used. Huffman encoding is a method of assigning variable-length bit strings to represent the characters in an alphabet such that frequent characters receive short representations while infrequent characters are assigned longer representations. Calculations were done on a list of the 10,000 most frequent words from Time magazine as a possible source for a dictionary. (For this experiment, each word occurred exactly once in the list.) Assuming even probabilities of the 26 letters, 4.7 bits are required per character. Noting the relative frequencies, the average information per character drops to only 4.2 bits. (Naturally, there are no such things in real computers as fractional bits, but the nature of Huffman encoding is that some characters receive encodings that are a little better than needed and others receive encodings that are a little long; our experiments show the overall inefficiency as being less than 1%.)

Still, there is residual redundancy. This is due to the fact that not all character combinations occur in proportion to the individual character frequencies. For example, "Q" is very infrequent, but, given that it occurs at all, it is almost certain to be followed by "U". On the other hand, "J" and "C" are both relatively frequent individually, but in our language, there are no words whatsoever in which an initial "J" is followed by "C". Intuitively, it should be possible to invent a new alphabet, some characters of which would represent a combination of several of our old characters.

This problem has been looked at in the past, but solutions have been attempted by exhaustive examinations of all

26 letters,
676 digrams (two-character combinations),
17,576 trigrams,
456,976 quadrigrams, etc.

Generally, an attempt is made to choose the most frequent combinations from each class and to then fine tune the selections by trial and error. However, there are two serious errors in this approach:

1) The computer searches are enormous and will still not necessarily produce the best symbol set (since the longest symbol is no longer than 4 characters - that being the longest sequence for which frequency tabulations are generally attempted).

2) It is incorrect to assume that because a combination is frequent, it will be a good symbol choice.

It is not the absolute frequency of a letter combination which makes it significant - it is its departure from expected frequency which is important. This departure is really a way of noting that English spelling is a Markov process - the probability that the next character will be a "P", for example, depends on what the last character was.

It is possible to calculate the effect of adding a particular digram symbol to the English alphabet. (Ignore, as minor, the effects of the final Huffman coding inefficiency.)

Let

T = total number of characters needed to encode a message with the latest alphabet.

c_i = number of occurrences of character i.

c_j = number of occurrences of character j.

$c_{i,j}$ = number of occurrences of character i followed by character j.

$B_{i,j}$ = bits to be saved by encoding a digram of character i followed by character j.

= (bits used with latest alphabet) - (bits used with new digram added)

Momentarily ignoring the effect of the new digram on the other

(non-participating) characters,

$$B_{i,j} = -(C_i \log_2(c_i/T) + c_j \log_2(c_j/T)) + ((c_i - c_{i,j}) \log_2((c_i - c_{i,j})/(T - c_{i,j})) + (C_j - c_{i,j}) \log_2((c_j - c_{i,j})/(T - c_{i,j})) + c_{i,j} \log_2(c_{i,j}/(T - c_{i,j})))$$

There is also a savings on the non-participating characters; their frequencies increase slightly since fewer characters are used overall, and thus this must be accounted for. For each non-participating character, savings to be gained may be calculated.

Let

c_k = number of occurrences of non-participating character k.

B_k = coding improvement over all the occurrences of character k due to the digram.

$$= -c_k \log_2(c_k/T) + c_k \log_2(c_k/(T - c_{i,j}))$$

$$= c_k \log_2(c_k T/c_k(T - c_{i,j}))$$

$$= c_k \log_2(T/(T - c_{i,j}))$$

Summing over all "k" characters not participating, the total savings on non-participating characters can be found.

Let

$N_{i,j}$ = Total savings on all non-participating characters due to digram of characters i and j.

$$= (T - 2c_{i,j}) \log_2(T/(T - c_{i,j}))$$

The total savings due to the addition of the digram symbol to the alphabet is the sum of these components:

$$B = \text{total savings} = B_{i,j} + N_{i,j}$$

There is one final complication. If characters i and j are the same, then the formula for $B_{i,j}$ must be altered so as to avoid "charging double" for encoding a single character. In this special case only,

$$B_{i,i} = - c_i \log_2 (c_i/T) + (C_i - c_{i,i}) \log_2 ((c_i - c_{i,i})/(T - c_{i,i}))$$

To build up a new alphabet, new digram symbols are added one at a time. After each new symbol is added, all possible digrams are re-examined to find the next reduction. Each new digram reduction may be viewed as another stage in the compression machine. Some symbols may be used only in intermediate stages and may never be present at the final output due to later digram consolidation with other symbols, but this is immaterial.

Using these formulas, the best choice as the next character to be added to a symbol set in order to reduce redundancy can easily be found (on the computer). This was done in a PL/I Optimizer program running on the same list of the 10,000 most frequent words in Time. The results were interesting in that some very long strings were reduced to single symbols. For example, the 42nd symbol was "ATION", the 54th was "IGHT", the 105th was "COUNT", the 229th was "VOLUTION", etc. By running the program until an alphabet of 255 symbols had been created, the number of bits per character (referring to the "original" characters) had been reduced to only 3.77. Recounting our results, we find:

4.7 bits/character before any redundancy eliminated,

4.2 bits/character after accounting for relative character frequencies, and

3.77 bits/character after digram compression.

To use digram compression, one must have a method which insures that

1) the resulting "spelling" will be unique for a given message, and

2) the translation will be fast.

The first requirement is imposed by the fact that if, for example, the word "going" were to be compressed using an "alphabet" containing "oi", "ng" and "ing" characters, one would not (without some rules) know whether to encode

going = g + oi + ng

or

going = g + o + ing.

Any encoding must be done in the same way that characters were added to the alphabet. In our example, if the order in which the digrams were added was "ng", "oi", "ing", the first spelling would be correct; if the order were "ng", "ing", "oi", the second would be used.

The encoding is done by viewing the digram symbols as numbered compression rules, each rule specifying that at a given "step", a certain character pair may be replaced by a digram symbol. Particular rules may or may not apply to a given message (for example, a rule joining "N" and "G" does not apply if there is no "NG" pair); any rules that apply must be used in the order specified by their numbers.

The procedure for encoding a message is to mark each character pair with a rule number and the resulting symbol, if a rule exists for merging the two characters. The rule with the lowest number is performed. The compression possibilities are unchanged for all characters except the new digram symbol and the character which precedes it, and the digram and its following character. Lookups are done to find compression rules for these two character pairs only, and the process is repeated until no more compressions can be done. The symbols are then encoded into the varying length bit strings using Huffman coding for the compression alphabet.