Artificial Intelligence and Language Processing
C.A. Montgomery Editor

Experiments in Text File Compression

Frank Rubin
IBM, Poughkeepsie

A system for the compression of data files, viewed as strings of characters, is presented. The method is general, and applies equally well to English, to PL/I, or to digital data. The system consists of an encoder, an analysis program, and a decoder. Two algorithms for encoding a string differ slightly from earlier proposals. The analysis program attempts to find an optimal set of codes for representing substrings of the file. Four new algorithms for this operation are described and compared. Various parameters in the algorithms are optimized to obtain a high degree of compression for sample texts.

Key Words and Phrases: text compression, data file compaction, Huffman codes, N-gram encoding, comparison of algorithms

CR Categories: 3.7, 3.73, 4.33

1. Introduction

Let a fixed body of text be given. The text consists of a string of characters in some representation such as ASCII code. It is desired to transform this string into a new string containing the same information, but whose length is as small as possible. This will reduce storage costs and transmission costs. Encoding the data for security or for correction of transmission errors is not a primary concern here; however, the coding schemes proposed may be combined with data security or error correcting codes.

The principal goal here is to achieve the greatest possible degree of compression. The algorithms that are presented here are all computationally feasible, and require internal computer storage easily available on today's larger computers (computation cost, however, has generally been a secondary concern). Further, a technique is sought which will be applicable to a broad range of texts: examples are English prose, prose in any other language, source code in any computer language, object code, digital photographic images, telemetry data, digitized encephalograph or cardiogram data, digitized speech, and business files such as inventory, mailing lists, or order files.

The basic approach to compression used here is to divide the text string into substrings, and to replace each substring by a code. The simplest such scheme is to choose strings of identical characters [5] and replace them by a repetition count. This is well suited to source programs with long blank strings, business data with long zero strings, or to digital images with regions of uniform darkness. However, it allows the encoding of only a small portion of the possible substrings in any file.

Another simple approach is to choose substrings, typically character pairs, on the basis of standard English character frequencies [4, 5]. This is suitable for English prose, and also gives some compression for computer source languages, foreign language prose, or name and address data. The restriction to two-character substrings severely limits the degree of compression which can be obtained.

A third approach is to choose words [6, 7] or phrases [2] as the units of compression. Again, this limits the choices of substrings for which codes can be substituted. It is suited only to prose files and (slightly) to source program data.

2. Definitions

A few terms are needed to define the problem precisely. The body of text to be encoded will be called the *input string*. The semantic units of the input string will be called *characters*, and may consist of Roman letters, light intensities, or pulse amplitudes. The codes which represent the characters will be called *input codes*, and the correspondence between the input characters and the input codes will be called the *input representation*. The input representation might be paper tape code, grayness level codes, or decimal digit strings.

Encoding consists of dividing the input string into substrings called *input groups*. Each input group is replaced by an *output code*. The correspondence between input groups and output codes is called the *output representation*.

Author's address: IBM, Department E36, Building 707, Poughkeepsie, NY, 12602.

The goal of encoding is to obtain as compressed an output as possible. One could simply let the output code 0 represent the input group consisting of the entire input string. Obviously, there is no gain with this procedure, as it would still be necessary to show that this code corresponded to that particular input group. The length of the output is therefore regarded as including both the string of output codes, and the output representation. The measure of compression which is used is thus:

(len (input string) − [len (output string) + len (output rep)])
/len (input)

expressed as a percentage.

3. The Problem

There are two basic problems involved in finding a compact encoding of the input string. First the output representation must be determined, and then the division of the input strings into groups. The former problem itself contains two parts: determining the set of input groups, and determining the corresponding output codes.

The basic approach to the problem is similar to that of McCarthy [3]. The system consists of three parts: an analysis program which determines the output representation, an encoder which divides the input string into groups and substitutes the output codes, and a decoder which replaces output codes by the corresponding input groups. For comparatively short texts of fixed content, the analysis program is applied to the entire text. For longer texts, a statistically significant sample of the text is subjected to the analysis program. For texts whose contents are periodically changed or increased, the analysis program may be applied at intervals to vary the output representation as the statistical properties of the text file change. Since this requires decoding and then re-encoding the entire body of text, it should not be performed too frequently.

4. Encoding Algorithms

For many applications, the simple encoding scheme below will be preferred. It is not always optimal, but will be optimal in many cases, for example, when all input groups have equal length.

Algorithm A (Simple encoding)

1. Set the output string empty.
2. Choose the longest initial substring of the input string which is an input group.
3. Delete this initial substring from the input string, and append the corresponding output code to the end of the output string.
4. If the input string is not empty, repeat from step 2. Otherwise stop.

That this procedure is not always optimal can be seen by the following example. Let the input string be the English word ARGUMENT. Let the output representation be (in part):

Input group:	AR	ARG	GU	UM	ME	EN	MENT	T
Output code:	1	2	3	4	5	6	7	8

Algorithm A yields the division ARG-UM-EN-T and output string 2468 with four output codes, whereas the division of the input string AR-GU-MENT produces a three code output string 137.

To achieve an optimal encoding for a given output representation, it is necessary to consider several partial codings simultaneously. Let these be $P_1, P_2, \ldots, P_m$, corresponding to the first $I_1, I_2, \ldots, I_m$ characters of the input string, respectively.

Algorithm B (Optimal encoding)

1. Set the output string empty, $m = 1$, $I_1 = 0$, and $P_1 =$ empty string.
2. Choose i, $1 \leqslant i \leqslant m$ such that I_i is minimum.
3. Let $x_1, x_2, \ldots, x_n$ be the possible input groups beginning at character $I_i + 1$ of the input string, and $y_1, \ldots, y_n$ their corresponding output codes.
   For $j = 1, 2, \ldots, n$ DO
   a. If $L = I_i + \text{len}(x_j) = I_k$ for some k, $1 \leqslant k \leqslant m$, then if $\text{len}(P_i) + \text{len}(y_j) < \text{len}(P_k)$ replace P_k by $P_i y_j$. Otherwise, ignore y_j.
   b. If $L \neq I_k$ for any k, append $P_i y_j$ to the partial coding list. Set $m = m + 1$.
4. Delete P_i from the partial list. Set $m = m - 1$.
5. If $m = 1$, then move P_1 into the output string, and set $P_1 =$ empty string.
6. If $\min(I_i) = \text{len(input string)}$, done. Otherwise, repeat from step 2.

Step 5 is not logically necessary to the algorithm. It is done to minimze the length of the implied moves in steps 3a and 3b.

5. Developing the Input Groups

A seemingly ideal way to develop the input groups would be to choose the most frequent sequence in the input string, substitute a code for each occurrence of this group, choose the next most frequent sequence in the recoded input string, and so forth. There are three objections to this procedure. (1) There are practical problems in counting the frequency of all sequences. For example, if there are 26 input characters, there are more than 26^{10} sequences of lengths 2 to 10 possible. (2) The procedure requires recoding the input string as many times as the number of input groups ultimately chosen. (3) The procedure is not near optimal. For example, if the input group QU is chosen at some stage, and subsequently the input groups QUA, QUE, QUI, and QUO are chosen, then the group QU is probably no longer useful, assuming English text.

McCarthy's technique [3] is fairly successful at overcoming problems 1 and 2. His approach is to take all sequences of L consecutive input codes, where L is a prespecified maximum input group length and to sort them. Every subsequence of length 2 to L occurs as an

initial string of some sequence of length L, except for those within the last $L-1$ characters of the text. The frequency of every sequence can thus be determined directly from this sorted list.

This technique requires recoding of the source string only when the sequence selected overlaps some other sequence so that the frequency of the second sequence can no longer be determined from the sorted list. Otherwise, McCarthy recodes the sorted list itself to adjust the frequencies. This may, in fact, be a false economy. Although it takes some time to perform an optimal recoding of the entire input string, the sorted list is approximately $L-1$ times as long as the input string. (It may be somewhat shorter if equal elements in the sorted list are combined, but only a small percentage of such equal elements is likely if L is even moderately large, say $L \geqslant 8$.)

To make the text compression techniques feasible for the large samples which may be required with fairly heterogeneous texts, some techniques not requiring frequent recoding of the input string seem desirable. Two such new techniques will be presented here, based upon the concept of accretion. This simple idea is that an input group $a_1a_2 \ldots a_p$ can be built up in stages, for example, a_1a_2, $a_1a_2a_3$, . . . , $a_1a_2 \ldots a_p$. Obviously, if s and t are sequences, $freq(s) \geqslant freq(st)$ and $freq(t) \geqslant freq(st)$. This suggests that the set of sequences of length up to L can be built up in an $L-1$ stage process. This, however, ignores problem 3 above, namely that later sequences make earlier choices unneeded. The following algorithm embodies these concepts:

Algorithm C (Accretion)

1. Let the input groups be the single characters. Encode the input string.
2. Count the frequency of all input groups and augmented input groups in the encoded string.
3. Choose the best set of input groups and augmented input groups for the new set of input groups.
4. Encode the input string with the new set of input groups (Algorithm B).
5. If the length of the output string or the number of input groups has changed, and the maximum number of cycles has not been reached, repeat from step 2.

6. Accretive Group Development

Two aspects of Algorithm C have deliberately been left open. First, the augmentation of the input group may logically occur in any of three ways: (a) append a single input character to the left end of the group, (b) append a single character to the right end of the group, or (c) combine two complete groups into a new input group. Second, the definition of "best" group has been left open. It is not a priori obvious that the most frequent groups are the most valuable, and it may be desirable to take the length of the group into account. That is, a frequent long group would be considered more valuable than a frequent short group.

To settle such questions, three sample texts were encoded by Algorithm C, using each of the above alternatives. (Copies of these sample texts may be obtained from the author.) The three texts had the following characteristics:

Text 1. Prose English in upper and lower case, containing imbedded editing codes. The text included a large number of mathematical equations and computer instruction mnemonics in upper case. Total length was 29,305 characters, with 85 distinct characters present. The file consisted of variable length records with trailing blanks deleted.

Text 2. A PL/I source program, heavily commented, all upper case. The file contained fixed length 80 character records with serialization in positions 73 to 80. Total length was 35,040 characters, with 56 distinct characters represented.

Text 3. A 256×256 digitized photographic image, with 16 levels of gray scale.

The three sample texts were encoded by Algorithm C. The number of input groups which were allowed was 256, including the input groups consisting of a single character. The choice of best sequence was based entirely upon maximum frequency, subject to condition $F+L+1 < LF$ where L is the length, and F the frequency of the input group. Here LF is the total number of characters in all occurrences of the group, and $F+L+1$ the number of codes used if all occurrences of the input group receive the new substitute. Both input and output codes were 8-bit bytes. Up to 20 cycles or phases of the augmentation procedure were allowed.

Results of this test are shown in Table I. The upper figure in each cell is the number of output codes produced. The middle figure is the number of bytes used to represent the input groups. One byte was used for each single character group, and $L+1$ bytes were used for a group of $L > 1$ bytes. The lower figure is the net reduction in text length.

Comparing the figures in each column, it is clear that combining pairs of input groups gives a greater degree of compression than either method of appending single characters. This is because combining a pair of input groups always causes a decrease in the number of output codes, whereas appending a single character to an input group may simply subtract that character from the adjacent group. In particular, adding the character to the right end of the input group "beheads" the following group and thus may require the development of a new successor group before the augmented group can be used.

6.1 Measures of Savings

The second open question in the accretion technique is what measure of "best" input group should be used. The measure should reflect the amount of savings obtained by using that particular input group. Certainly, frequency is directly related to the degree of savings,

Table I. Comparison of accretion methods. In each cell, the upper figure is the length of the output string in bytes, the middle figure is the length of the output representation, and the lower figure is the percentage of compression achieved.

Method	Text 1 29305	Text 2 35040	Text 3 65536
Append characters to the left end of group	13686 787 50.6%	8632 1096 72.2%	29156 882 54.1%
Append characters to the right end of group	15309 708 45.4%	9840 918 69.2%	29696 870 53.3%
Combine pairs of input groups	12756 827 53.7%	8015 1183 73.7%	27211 947 57.0%

Table II. Variations on the accretion method for developing input groups. In each cell, the upper figure is the length of the output string in bytes, the middle figure is the length of the output representation, and the lower figure is the percentage of compression achieved.

Method	Text 1 29305	Text 2 35040	Text 3 65536
Using F-L as measure of savings	13232 804 52.0%	8630 984 72.5%	27016 952 57.3%
Using $F^*(L-1)$ as measure of savings	13054 1390 50.7%	8781 1910 69.4%	26666 1751 56.6%
Shortcut (using simple encoding until last phase.)	13398 773 51.7%	8667 1118 72.0%	28202 900 55.5%
Extended codes, 511 input groups	10780 2494 54.6%	7097 3219 70.5%	22955 2891 60.5%
512 9-bit codes	10294 2196 57.3%	7780 2411 70.9%	21292 2708 63.3%

and therefore the number of times a group appears in the input string is an obvious measure to use. If the group were created by combining two groups A and B of lengths L_A and L_B, then the storage required before substitution will be $2F + (L_A + 1) + (L_B + 1)$ and after substitution will be $F + (L_A + L_B + 1)$. The savings, therefore, will be $F + 1$ if groups A and B are subsequently eliminated, or $F - (L + 1)$ if groups A and B are subsequently retained. This justifies the use of frequency as a plausible measure of savings. It also suggests $F - L$ as a possible alternative measure.

A second, more naive measure of savings would be to compare the number of codes, FL, to represent the input groups without encoding to the $F + L + 1$ codes needed to represent F occurrences as single output codes. The savings would then be $F(L - 1) - 1$ codes.

Using the preferred method of combining complete pairs of input groups, accretion was performed to maximize the measures of savings $F - L$ and $F(L - 1)$. The results are presented in Table II and may be compared to the results for maximizing frequency in Table I. The encodings for $F - L$ are slightly less compact than for F. As may be expected, the input group tables were larger for $F(L - 1)$ because this measure tends to choose longer input groups. Further, because this measure omits many frequent sequences, the encoded strings are also longer. From these results, frequency is confirmed as the best measure of savings in choosing input groups.

6.2 Variations on the Accretion Method

An attempt to shortcut the accretion process to reduce computation cost, was done by using the simple encoding scheme (Algorithm A) for accretion up to the point of convergence, and then re-encoding the string optimally (Algorithm B) using the input groups developed. These results are also shown in Table II, and should be compared to the results in Table I. It is clear that use of the simple encoding scheme makes an inferior choice of input groups.

Another variation on the accretion technique is to attempt to extend the number of input groups beyond the limit imposed by the radix of the host computer. For 8-bit binary machines this limit is 256. One method is to use 9-bit codes. However, to preserve the byte orientation of the technique, a second form of extension was also tried. One code among the 256 possible was reserved as an extension code. When this code appeared in the first byte of an output code, the output code was then assumed to be two bytes. This gives a total of 255 one-byte output codes and 256 two-byte output codes.

At first this might seem hopeless. A new input group is formed by concatenating two shorter input groups. However, it is also possible that in several phases three or more shorter input groups will be combined. Also, the extended set of groups could cause some groups to be subsumed. For example, in English text, if input groups ANG, ING, and ONG are chosen, then group NG might be deleted. This will cause the most frequent extended code to be promoted from a 2-byte to a 1-byte code, and allow the creation of one more input group. This variation was tested, with the results shown in Table II. It results in a significant improvement for Texts 1 and 3, but by accepting too many groups, it is significantly poorer for Text 2 than the encoding with 256 groups.

The alternative method for extending the number of groups is to use 9-bit output codes instead of 8-bit codes. A full 512 input groups may then be used. If a candidate pair for combination appears F times, it will require $18F$ bits before combining and $9F + 8(L + 1)$ bits after combining. Therefore, to trim the set of input groups, we reject any group for which $18F \leqslant 9F + 8(L + 1)$ or $9F \leqslant 8(L + 1)$. This produces a strong bias towards shorter input groups, as can be seen by

comparing the sizes of the output representations in the last two rows of Table II. For all three texts, the 9-bit codes produce better compression than the extended codes described above. However, for Text 2, far too many groups were accepted, so that the 512 group encoding is still less effective than the 256 group encoding.

7. Huffman Encoding

Huffman codes [1] are variable length codes in which the number of bits used to encode a message (input group) will be inversely proportional to the logarithm of its frequency. Since the most frequent input groups have short codes, the total number of bits needed to represent the entire text string is reduced.

The basic idea is to use Algorithm C to select a set of input groups, and then to provide a Huffman encoding for these groups based upon their frequency in an optimal encoding of the input string. This is Algorithm D.

The basic problem with the application of Algorithm D is determining the number of input groups to use. Merely to get a feel for the problem, three numbers of groups were tried, NSING, NSING + 100, and 512. NSING is the number of single characters represented in the input string. These results are presented in Table III. It is clear that the method is powerful, but for Text 2 that 512 is too many groups, That is, some unprofitable groups were included. To remedy this, two measures of profitability will be developed.

Consider first the storage needed to represent an input group. We need the group itself, L bytes, the length, the Huffman code, and the length of the Huffman code. For texts of the size considered here, the Huffman codes cannot exceed 16 bits. Since input groups were artificially restricted to 12 characters, each length can be coded in 4 bits. Allowing the two lengths to share an 8-bit byte, $L + 3$ bytes or $8L + 24$ bits are sufficient.

Now if the frequency or number of occurrences of an input group is F in the optimal encoding of the input string, and the length of the encoded input string is N, then the number of bits required to represent that input group is about $B = \lceil \log_2 N/F \rceil$ where $\lceil x \rceil$ represents the least integer not less than x. Before combining pairs, let F_1 and F_2 be the frequencies of the component pairs, $B_1 = \log_2 N/F_1$, and $B_2 = \log_2 N/F_2$. The storage for the combined pair will be $FB + 8(L + 3)$ bits, compared to $F(B_1 + B_2)$ bits before combining. Assuming that both component groups are retained, therefore, we require that $FB + 8(L + 3) < F(B_1 + B_2)$ for any candidate pair. McCarthy [3] claims that it is sufficiently accurate to assume that each component code is one byte. Therefore, we also try $FB + 8(L + 3) < 16F$ as a possible test for a profitable group.

These results are also presented in Table III. It is obvious from comparing the table size 4138 for Text 2, using 512 groups to the 2874 and 2917 using the above profit criteria, that many unprofitable groups have been trimmed. However, it is equally clear that the second measure eliminates some profitable groups as well. For Texts 1 and 3 the second measure also eliminates some profitable groups. In the case of Text 1, some of the groups eliminated by the first measure are also profitable. It is seen, then, that McCarthy's approximation is not at all satisfactory, while measure 1 is only partially successful.

Table III. Comparison of Huffman encoding compression for different input group sets. In each cell the upper figure is the length of the output string in bytes, the middle figure is the length of the output representation, and the lower figure is the percentage of compression achieved.

Group Set	Text 1 29305	Text 2 35040	Text 3 65536
Single characters	17105	13133	32768
	340	224	64
	40.4%	61.8%	49.8%
Single characters + 100 most frequent	13993	8319	30570
	926	951	566
	49.0%	73.5%	52.4%
512 most frequent	9579	6093	20904
	3525	4138	3733
	55.2%	70.7%	62.4%
$FB + 8(L + 3) < F(B_1 + B_2)$	10320	7115	20811
	3172	2874	3697
	53.9%	71.4%	62.6%
$FB + 8(L + 3) < 16F$	11230	7830	21330
	2987	2917	3626
	51.4%	69.3%	61.9%

8. Incremental Encoding

Thus far we have used the accretive method as the sole means of finding the input groups to which to apply Huffman encoding. A second method may be based upon the concept of incremental encoding. Assume that we are choosing the input groups one at a time. Ideally, we would like to choose the next group as the one which makes the greatest savings. Retaining from the accretion method the basic concept that a new group is formed by combining two older groups, the amount of savings which can be achieved by combining any pair can be estimated from its frequency and the frequency of its components.

It is certainly not feasible to make a pair frequency count at every stage. However, the frequency count can be updated from the previous count at each stage. For instance, let the pair be BC and assume the context $ABCD$. Then set $freq(AB) = freq(AB) - 1, freq(AX) = freq\ (AX) + 1, freq\ (BC) = 0, freq\ (CD) = freq\ (CD) - 1$, and $freq\ (XD) = freq\ (XD) + 1$, where X is the new code substituted for BC. This will be done for each

occurrence of BC in the coded output string, and every occurrence of BC will be replaced by X.

The process of combining groups will continue until the maximum number of groups has been reached, or until no more profitable substitutions can be made. At this stage, two problems occur. First, some of the groups chosen earlier may have been subsumed, and second, the encoding thus achieved may not be optimal for the set of input groups developed. The following algorithm deals with these problems.

Algorithm E (Incremental encoding)

1. Encode the single characters of the input string and count code pairs.
2. Select the most profitable code pair. If no profitable pairs exist, then skip to step 6.
3. Replace all occurrences of the pair by the new pair code.
4. Adjust the pair frequency tables.
5. If any unassigned codes remain, repeat from step 2.
6. If no new pairs were combined in steps 2 to 4 since the last repetition of step 7, then stop.
7. Optimally encode the input string using the current input group set.
8. Count the code frequencies.
9. Delete any code of frequency 0.
10. If any code has frequency 1, delete all such codes and repeat from step 7.
11. Repeat from step 2.

The principal question to be answered for this method is the number of input groups to use. McCarthy [3] suggests a cutoff on the basis of frequency. An input group is rejected if its frequency is less than $N/1500$, where N is the length of the input string. The importance of making this cutoff can be seen by comparing the sizes of the output representation in the first two rows of Table IV. The cutoff for $F \leqslant 1$ is essentially no cutoff, so that the output representation becomes very large.

The compression is greater for Texts 1 and 3 without the cutoff suggested by McCarthy than with it. This indicates that the cutoff frequencies, which were 19 and 43, may be too high for these texts. Consequently, a cutoff of $F \leqslant N/3000$ was also tried. This increased the compression for all of the texts. Since further increase of K in the cutoff of N/K would set the cutoff too low for Text 1, a constant cutoff of $F \leqslant 7$ was also tried. This somewhat increased the compression for Texts 1 and 3, but decreased compression for Text 2. Further minor adjustments of the cutoff produced differences too small to be considered significant. The ideal cutoff appears, from these results, not to be any fixed fraction of the text length, but a constant on the order of 7 to 9.

By comparing the last row of Table II to the last row of Table IV, it is seen that the contribution of Huffman encoding to the degree of text compression achieved is not impressive. This suggests that the incremental method of encoding might well be tried by itself. Several experiments along this line were performed, with the results shown in Table V. In each case (up to) 512 9-bit codes were used. In the first two experiments, frequency was used as a measure of savings. Groups with frequency not greater than the cutoff of 1 and 7, respectively, were deleted. To reduce execution time, recoding (step 10 of Algorithm E) was performed only once after deletion (step 9). In the third experiment $F - L$ was used as the measure of savings, with an input group deleted whenever $F - L \leqslant 1$. This latter measure was slightly better.

A difficulty with the incremental method of developing the input groups is that some of the earlier groups may be largely subsumed by groups developed later. To

Table IV. Comparison of Huffman encoding compression using incremental group development with different frequency cutoffs. In each cell, the upper figure is the length of the output string in bytes, the middle figure is the length of the output representation, and the lower figure is the percentage of compression achieved.

Cutoff	Text 1 29305	Text 2 35040	Text 3 65536
$F \leqslant 1$	9654	6070	20731
	3448	3965	3676
	55.2%	71.3%	62.7%
$F \leqslant N/1500$	11821	8458	25055
	1419	848	1432
	54.8%	73.4%	59.5%
$F \leqslant N/3000$	10258	7358	21507
	2489	1556	2808
	56.4%	74.5%	62.8%
$F \leqslant 7$	9691	6740	20214
	3002	2371	3707
	56.6%	73.9%	63.4%

Table V. Variations on the incremental method for developing input groups. In each cell, the upper figure is the length of the output string in bytes, the middle figure is the length of the output representation, and the lower figure is the percentage of compression achieved.

Method	Text 1 29305	Text 2 35040	Text 3 65536
Maximize F	10278	6822	21429
Cutoff for $F \leqslant 1$	2344	2905	2618
	56.9%	72.2%	63.3%
Maximize F	10278	8008	21429
Cutoff for $F \leqslant 7$	2344	1684	2618
	56.9%	72.3%	63.3%
Maximize $F - L$	10363	7181	21310
Cutoff for $F \leqslant L + 1$	2130	2235	2501
	57.3%	73.1%	63.6%
Maximize $F - L$	10129	7591	20820
Cutoff for $F - L \leqslant 10, 4, 2, 1, \ldots$	2143	1774	2566
	58.1%	73.2%	64.3%
Maximize $F - L$	10141	7377	20799
Cutoff for $F - L \leqslant 1$	2144	1839	2557
Delete for $F - L \leqslant 10, 4, 2, 1, \ldots$	58.0%	73.6%	64.3%

expand on an earlier example, suppose NG is an early group in English text, and that ANG, ING, and ONG are developed later. There may still be a few occurrences of ENG or UNG, so that the group NG is retained. But there may be other groups of greater value than NG.

An attempt to deal with this problem was made by manipulating the cutoff level. After the first iteration, the cutoff is set fairly high. This tends to eliminate many of the subsumed input groups. On each subsequent iteration, the cutoff is reduced until it reaches 1. For the experiments here, the cutoff was initially set to 10, and subsequently *CUTOFF* was set to (*CUTOFF* + 2)/3. Two variations of this procedure were used, with results presented in rows 4 and 5 of Table V. In the first, faster variant, selection of input groups was terminated for groups with $F - L \leqslant CUTOFF$, and, after optimal recoding, any group for which $F - L \leqslant CUTOFF$ was deleted. In the second variant, selection of input groups was allowed to run to completion. That is, up to 512 groups for which $F - L > 1$ were generated. Then, after optimal recoding, any group for which $F - L \leqslant CUTOFF$ was deleted. Both of these methods proved to be more effective than any of the other variants of the incremental encoding algorithm with fixed cutoff levels. Several other values for the cutoff levels were tried, with results not significantly different from the levels just mentioned.

The above algorithms may be regarded as cases of the algorithm following.

Algorithm F (Shortcut variable-cutoff-level incremental encoding)

1. Encode the input string as single character groups. Set the initial cutoff and deletion levels.
2. While there are code pairs with *F–L* greater than the cutoff level repeat steps a, b, and c.
   a. Select the most frequent code pair. Form a new input group.
   b. Replace all occurrences of the pair with the output code for the new group.
   c. Adjust the code pair frequency table.
3. If no new code pairs were selected in step 2, then
   a. If the cutoff and deletion levels are at their final values, then stop.
   b. Otherwise skip to step 5.
4. Optimally encode the input string using the current set of input groups.
5. Delete any input group whose corresponding output code appears fewer times in the encoded string than the group length plus the deletion level.
6. If any groups were deleted in step 5, then optimally recode the input string.
7. Adjust the cutoff and deletion levels. Repeat from step 2.

9. Conclusions

A high degree of text compression can be obtained with either of two iterative schemes, Algorithm C or Algorithm F. Under Algorithm C, it is desirable to maximize the frequency of the input groups chosen. Under Algorithm F, it is desirable to maximize frequency minus length. Huffman encoding cannot be combined profitably with these group encoding schemes because the output representation requires more characters to specify. The process of combining characters into groups produces a fairly even frequency distribution which defeats the purpose of the Huffman encoding. The most compact Huffman encoding was obtained by using all input groups whose frequency was at least a certain fixed integer, in this case 8, rather than a fixed fraction of the string length.

The methods of this paper have a fairly high computation time and large storage requirements. Further work will be needed to reduce these. The methods may be applied to slowly changing bodies of stored text periodically to reduce computation cost. For applications with fixed statistical text properties, such as standard English prose, a single application of the analysis procedures would be sufficient.

Received March 1975; revised September 1975

References

1. Huffman, D.A. A method for the construction of minimum redundancy codes. *Proc. IRE 40* (Sept. 1952), 1098.
2. Wagner, R.A. Common phrases and minimum-space text storage. *Comm. ACM 16*, 3 (March 1973), 148–152.
3. McCarthy, J.P. Automatic File Compression. International Computing Symp. 1973, North-Holland, Amsterdam, 1974, pp. 511–516.
4. Snyderman, M., and Hunt, B. The myraid virtues of text compaction. *Datamation 16*, 12 (Dec. 1970), 36–40.
5. Ruth, S.S., and Kreuzer, P.J. Data compression for large business files. *Datamation 18*, 9 (Sept. 1972), 62–66.
6. Kusmiss, J.M. An experiment in adaptive encoding. IBM Tech. Rep. TR 00.2524, Poughkeepsie, N.Y., 1974.
7. Hagamen, W.D., et al. Encoding verbal information as unique numbers. *IBM Systems J. 11* (Oct. 1972), 278–315.

Corrigendum
Computer Systems

Forest Baskett and Allen Jay Smith, "Interference in Multiprocessor Computer Systems with Interleaved Memory," *Comm. ACM 19*, 6 (June 1976), pp. 327–334.

Page 331: Equation (13) should read

$$\frac{N}{M} = \frac{A''(1) + 2A'(1) - 2A'^2(1)}{2(1 - A'(1))^2}(1 - p)$$

Page 331: the equation two equations down from eq. (13) should read

$$A(Z) = (1 - p + pz)^n$$