J. J. Rissanen

Generalized Kraft Inequality and Arithmetic Coding

Abstract: Algorithms for encoding and decoding finite strings over a finite alphabet are described. The coding operations are arithmetic involving rational numbers l_i as parameters such that $\Sigma_i 2^{-l_i} \leq 2^{-\epsilon}$. This coding technique requires no blocking, and the per-symbol length of the encoded string approaches the associated entropy within ϵ. The coding speed is comparable to that of conventional coding methods.

Introduction

The optimal conventional instantaneous Huffman code [1] for an independent information source with symbol probabilities $(p_1, \cdots, p_m)$ may be viewed as a solution to the integer programming problem: Find m natural numbers l_i as lengths of binary code words such that $\Sigma_i p_i l_i$ is minimized under the constraining Kraft inequality

$$\sum_i 2^{-l_i} \leq 1.$$

Then the minimized sum $\Sigma_i p_i l_i$ approximates the Shannon-Boltzmann entropy function $H(p_1, \cdots, p_m) = -\Sigma_i p_i \log p_i$ from above with an error no more than one. If a better approximation is required, blocking is needed; e.g., a kth extension of the alphabet must be encoded, which reduces the least upper bound of the error to $1/k$ [2].

We describe another coding technique in which m positive *rational* numbers $l_1, \cdots, l_m$ are selected such that a generalized Kraft inequality holds:

$$\sum_i 2^{-l_i} \leq 2^{-\epsilon}, \qquad \epsilon > 0.$$

(For $\epsilon = 0$, the rationality requirement would have to be relaxed.) The length of the code of a sequence s with n_i occurrences of the ith symbol is given by the sum

$$L(s) = \sum_i n_i l_i,$$

which when minimized over the l_i subject to the preceding inequality and divided by $n = \Sigma_i n_i$ is never larger than

$$H\left(\frac{n_1}{n}, \cdots, \frac{n_m}{n}\right) + \epsilon + O\left(\frac{1}{n}\right),$$

where ϵ is determined by the difference $l_i - \log(n/n_i)$. This means that, if the strings are generated by an independent information source, the mean of $n^{-1}L(s)$ approaches the entropy function from above within an error $\epsilon + O(1/n)$.

The coding operations are arithmetic, and they resemble the concatenation operations in conventional coding in that the code of the string sa_k, where a_k is a new symbol, is obtained by replacing only a few (always less than some fixed number) of the left-most bits of the code representing s by a new longer string. As a result, the coding operations can be accomplished with a speed comparable to that of conventional coding.

The primary advantage of the resulting "arithmetic coding" is that there is no blocking needed even for the binary alphabet. In addition, for small alphabet sizes the size of tables to be stored and searched is smaller than the code word tables in conventional coding methods. For a binary alphabet a special choice of parameters l_1 and l_2 leads to a particularly simple coding technique that is closely related to one due to Fano [3].

The coding method described here is reminiscent of the enumerative coding techniques of Schalkwijk [4] and Cover [5], and perhaps also of that due to Elias, as sketched in [2]. All of these are inferior, however, in one or more crucial respects, especially speed.

Binary alphabet

The coding algorithm is derived and studied for any finite alphabet, including the binary case. But because of the special nature of a binary alphabet, which admits certain simplifications, we study it separately. The importance of applications of the binary alphabet in data storage also warrants separate study.

Let l_1 and l_2 be two positive rational numbers such that $l_1 \le l_2$ and when written in binary notation they have q binary digits in their fractional part. Further, for x, a rational number, $0 \le x < 1$, with q binary digits in its fractional part, let $e(x)$ be a rational-valued approximation to 2^x such that

$$e(x) = 2^{x+\delta_x}, \frac{\epsilon}{3} \ge \delta_x \ge 0, \qquad (1)$$

and that $e(x)$ has $r > 0$ binary digits in its fractional part. Clearly, the minimum size for r depends on ϵ and q. For a choice of these, see Theorem 2.

Let s denote a binary string and λ an empty string. Write the concatenation of s and k, $k = 0$ or 1, as sk. Define the rational-valued function L by the recursion

$$L(sk) = L(s) + l_{k+1},$$
$$L(\lambda) = r. \qquad (2)$$

Write the numbers $L(s)$ as

$$L(s) = y(s) + x(s), \qquad (3)$$

where $y(s)$ is the integer part $\lfloor L(s) \rfloor$ and $x(s)$ is the fraction with q fractional bits.

The encoding function C transforms binary strings into nonnegative integers by the recursive formula

$$C(s0) = C(s)$$
$$C(s1) = C(s) + 2^{y(s1)}e[x(s1)]$$
$$C(\lambda) = 0. \qquad (4)$$

Because the code increases only when a 1 has been appended to the string, C takes a binary string s_m, described by the list $\langle k_1, \cdots, k_m \rangle$, where k_i denotes the position of its ith 1 from left, to the sum

$$s_m \leftrightarrow \langle k_1, \cdots, k_m \rangle \overset{C}{\rightarrow} \sum_{i=1}^{m} \Phi(s_i), \qquad (5)$$

where

$$\Phi(s_i) = 2^{y(s_i)}e[x(s_i)]$$
$$y(s_i) = \lfloor (k_i - i)l_1 + il_2 \rfloor + r$$
$$x(s_i) = (k_i - i)l_1 + il_2 + r - y(s_i). \qquad (6)$$

The code for s consists of the pair $\langle C(s), L(s) \rangle$, where $L(s)$ may be replaced by the length $n = |s|$ of s and m, the number of 1's in s. Decoding function D recovers s from right to left recursively as follows:

If $C(s) < 2^{y(s)}e[x(s)]$,

Then generate a 0, i.e., $s = s'0$; (7)

Else, generate a 1, i.e., $s = s'1$.

In addition, in the former case make

$$C(s') = C(s)$$
$$L(s') = L(s) - l_1, \qquad (8)$$

and in the latter case make

$$C(s') = C(s) - 2^{y(s)}e[x(s)],$$
$$L(s') = L(s) - l_2, \qquad (9)$$

to complete the recursion.

As a practical matter, the test in (7) is done by truncating $C(s)$ to its $r + 1$ left-most bits $\overline{C}(s)$ and writing

$$\overline{C}(s) = 2^{\alpha(s)}\beta(s),$$

where $1 \le \beta(s) < 2$ and $\alpha(s) = |C(s)| + 1$. Then the order inequality in (7) is equivalent to the lexicographic order inequality

$$\langle \alpha(s), \beta(s) \rangle < \langle y(s), e[x(s)] \rangle, \qquad (10)$$

with priority on the first component.

We next give a Kraft inequality type condition for the numbers l_1 and l_2, which guarantees a successful decoding.

Theorem 1 If for $\epsilon > 0$,

$$2^{-l_1} + 2^{-l_2} \le 2^{-\epsilon}, \qquad (11)$$

then $D\langle C(s), L(s) \rangle = s$ for all binary strings s.

Proof We have to prove that $s = s'1 \iff C(s) \ge 2^{y(s)} e[x(s)]$. By (4) the implication from left to right is clear. To prove the converse is equivalent to proving that for all strings s

$$C(s) < \Phi(s0) \triangleq 2^{y(s0)}e[x(s0)], \qquad (12)$$

because $C(s0) = C(s)$ by (4).

By (1)-(3),

$$2^{l_{j+1}-\epsilon/3}\Phi(s) \le \Phi(sj) \le 2^{l_{j+1}+\epsilon/3}\Phi(s), j = 0, 1, \qquad (13)$$

and, because by (11) $l_1 > \epsilon$ and $l_2 > \epsilon$,

$$\Phi(s) < \Phi(sj). \qquad (14)$$

We prove (12) by induction. For the induction base

$$C(\lambda) = 0 < 2^{y(0)}e[x(0)],$$

because $L(0) = l_1 + r > 0$. Assume then that (12) holds for all s with length $|s| \le n$.

Case 1 — $s' = s0$
By (4), (12), and (14), in turn,

$$C(s') = C(s) < \Phi(s0) = \Phi(s') < \Phi(s'0),$$

and (12) holds for all strings s with length $n + 1$ ending at 0.

Case 2 — $s' = s1$

By (4), (12), and (13), in turn,

$$C(s') = C(s) + \Phi(s1) < \Phi(s0) + \Phi(s1) \leq (2^{l_1+\epsilon/3} + 2^{l_2+\epsilon/3})\,\Phi(s).$$

Again by (13) twice

$$\Phi(s) \leq 2^{-l_2+\epsilon/3}\,\Phi(s1) \leq 2^{-l_2+\epsilon/3} \times 2^{-l_1+\epsilon/3}\Phi(s10),$$

and we have

$$C(s') < (2^{-l_1+\epsilon} + 2^{-l_2+\epsilon})\,\Phi(s'0).$$

By (11), finally,

$$C(s') < \Phi(s'0),$$

which completes the induction and the proof of the theorem.

Theorem 2 If s is a binary string of length n with m 1's, then

$$\log C(s) \leq m\,l_2 + (n-m)l_1 + r + 1 + \epsilon/3. \qquad (15)$$

With $\epsilon \leq \epsilon_1$, $\epsilon_2 \leq \epsilon + 2^{-q}$ and

$$l_1 = \log\frac{n}{n-m} + \epsilon_1,$$

$$l_2 = \log\frac{n}{m} + \epsilon_2,$$

inequality $(11)'$ holds and

$$\frac{1}{n}\log C(s) \leq H\left(\frac{m}{n}\right) + \epsilon + 2^{-q} + 0\left(\frac{r}{n}\right), \qquad (16)$$

where $H(p) = p\log p^{-1} + (1-p)\log(1-p)^{-1}$.

Proof By (1), (5), and (13),

$$C(s) \leq 2^{r+\epsilon/3}\sum_{i=1}^{m} 2^{(k_i-i)l_1+il_2}.$$

The sum is clearly at its maximum when $k_i = n - m + i$, and hence

$$C(s) \leq 2^{r+\epsilon/3} \times 2^{(n-m)l_1+ml_2} \times \frac{2^{l_2}}{2^{l_2}-1} = \left(1 + \frac{1}{2^{l_2}-1}\right) \times 2^{(n-m)l_1+ml_2+r+\epsilon/3}.$$

By (11) and from the fact that $l_2 \geq l_1$,

$$2^{-l_2} \leq 2^{-\epsilon-1} \text{ or } 2^{l_2} > 2.$$

Therefore, $1/(2^{l_2}-1) < 1$, and the claim (15) follows. The rest follows by a direct calculation.

Remark If the symbols 0 and 1 are generated by an independent or memoryless information source, then, because $E(m/n) = p$, inequality (16) holds also for the means of both sides.

Numerical considerations, example, and special case

We next study the addition in (9). From (4), (13), and (12),

$$C(s1) \geq \Phi(s1) \geq 2^{l_2-\epsilon/3}\Phi(s) \geq 2^{l_2-l_1-\epsilon}\Phi(s0) \geq 2^{l_2-l_1-\epsilon}C(s).$$

In cases of interest, where $l_1 < \frac{1}{2}$ and $l_2 > 2$, this gives

$$C(s1) \geq 2^{l_2-1}C(s). \qquad (17)$$

It then follows that never more than $r - \lfloor l_2 \rfloor + 2$ left-most bits of $C(s)$ in (9) need be replaced in order to get $C(s1)$. This means that the recursion (9) is reminiscent of conventional coding methods such as Huffman coding in which a new code word is appended to the string of the previous code words. Here, there is a small overlapping between the manipulated strings.

What then are the practical choices for q, ϵ, and r? Suppose that $p = (m/n) < \frac{1}{4}$, which are the values that provide worthwhile savings in data storage; $[H(\frac{1}{4}) \cong 0.81]$. Then ϵ should be smaller than, say, $p \cong l_1$. This makes q not smaller than $\log p^{-1} - 1 \cong l_2 - 1$. The function 2^x is such that (1) in general can be satisfied approximately with $r = q + 2$.

The function $e(x)$ may be tabulated for small or moderate values of q, or it may be calculated from the standard formula of the type

$$e(x) = (1 + b_1x + \cdots + b_kx^k)^2. \qquad (18)$$

With a table lookup for $e(x)$ the coding operations involve only two arithmetic operations each per bit with the version (7)-(10) and approximately $6(m/n)$ operations per bit when the formulas (5), (6) are used. In the latter case the decoding test (10) is not done for every bit but rather the k_i in $s_i \leftrightarrow \langle k_1, \cdots, k_i \rangle$ are solved as: k_i is the maximum for which

$$\langle \alpha(s_i), \beta(s_i) \rangle \geq \langle y(s_i), e[x(s_i)] \rangle. \qquad (19)$$

In fact,

$$k_i = \hat{k} \text{ or } \hat{k} + 1,$$

where $\hat{k} = \lceil l_1^{-1}[\alpha(s_i) + i(l_1 - l_2) - r] \rceil$ can be calculated recursively with two arithmetic operations. If $\log e(x)$ is also approximated by a table, then k_i can be calculated without the above ambiguity with three arithmetic operations from

$$k_i = \lfloor l_1^{-1}\{\alpha(s_i) + \log e[\beta(s_i)] + i(l_1 - l_2) - r\} \rfloor. \qquad (20)$$

Thus the coding operations can be done at a rate faster or comparable to that of Huffman coding. If the function $e(x)$ and its logarithm are tabulated, the sizes of the tables are of the order of p^{-1}, which gives the error $\epsilon \cong p$ as the "penalty" term in (16).

With Huffman coding the same penalty term is k^{-1} where k is the block size [2]. Therefore, in order to guarantee the same penalty term, we must put $k \cong \epsilon^{-1} \cong p^{-1}$, which gives $2^k \cong 2^{1/p}$ for the number of code words. In typical cases this is much larger than the number of entries p^{-1} in the table of $e(x)$ with the present coding method.

Example We now illustrate our coding technique by a simple example. Let $l_1 = 0.01$ and $l_2 = 11.11$, both numbers being in binary form. Then

$$2^{-l_1} + 2^{-l_2} \leq 0.915 < 2^{-0.128}$$

Hence, $\epsilon = 0.128$ will do, and we get from (1) $r = 4$. The function e is defined by the table

x	$e(x)$
0.00	1.0000
0.01	1.0011
0.10	1.0101
0.11	1.1011.

Take $s \leftrightarrow \langle 1, 3, 20, 22, 42 \rangle$. Then from (6) and the table above,

$$\begin{aligned}
\Phi(1, 1) &= 2^6 \times 1.1011 \\
\Phi(3, 2) &= 2^{10} \times 1.1011 \\
\Phi(20, 3) &= 2^{18} \times 1.0101 \\
\Phi(22, 4) &= 2^{22} \times 1.0101 \\
\Phi(42, 5) &= 2^{31} \times 1.0000.
\end{aligned}$$

By adding all these we get

$$f(s) = 10000000010110010100011100101100.$$

Because of the complications involved in blocking, Huffman coding is rarely used for a binary alphabet. Instead, binary strings are often coded by various other techniques, such as run-length coding in which runs of consecutive zeros are encoded as their number. We describe a special case of arithmetic coding that is particularly fast and simple. This method turns out to be related to one due to Fano [3], and the resulting compression is much better than in ordinary run-length coding.

Suppose that we wish to store m binary integers,

$$Z_m = \langle z_1, z_2, \cdots, z_m \rangle, 0 \leq z_1 \leq z_2 \leq \cdots \leq z_m \leq 2^w,$$

each written with w bits. Let $n = \lceil \log m \rceil$, and let $n < w$. The list Z_m may be encoded as a binary string s_m whose ith 1 is in position $k_i = z_i + i$, as in (5). Then p is approximately given by 2^{n-w}, and we may put

$$l_1 = 2^{n-w}, \qquad l_2 = w - n + 1.$$

Then inequality (11) can be shown to hold for some ϵ whose size we do not need. By setting $e(x) = 1 + x$ and $r = w - n$ we get from (6)

$$\begin{aligned}
y(s_i) &= q_i + i(r + 1) + r, \\
x(s_i) &= r_i 2^{-r}, \\
\Phi(s_i) &= (2^r + r_i) 2^{q_i + i(r+1)},
\end{aligned}$$

where r_i consists of the $w - n$ right-most bits of z_i and q_i of the rest; i.e.,

$$z_i = q_i 2^{w-n} + r_i.$$

The encoding of Z_m or, equivalently, of s_m is now straightforward by (5). It can be seen to be a binary string where the m strings r_i and q_i appear as follows

$$1 r_m 0 \cdots 0 \cdots 1 r_2 0 \cdots 0 1 r_1 0 \cdots 0,$$

where the first run of zeros counted from the right is of length q_1, the next of length $q_2 - q_1$, and so on; the last or the left-most run is of length $q_m - q_{m-1}$. From this the r_i and the q_i can be recovered in the evident manner. The length of the code is seen to be not greater than $m(w - n + 2)$, which is an excellent approximation to the optimum, $2^w H(2^{n-w})$, whenever $w - n > 3$.

Finite alphabet

Let $\langle a_1, a_2, \cdots, a_m \rangle$ be an ordered alphabet, and let ϵ be a positive number, which we will show determines the accuracy within which the optimum per symbol length, the entropy, is achieved. Let $l_1, \cdots, l_m$ be positive rational numbers with q binary digits in their fractional parts such that

$$\sum_{i=1}^{m} 2^{-l_i} \leq 2^{-\epsilon}. \tag{21}$$

Further, let p_k be a rational number such that

$$p_k = 2^{-l_k + \epsilon_k}, \frac{2\epsilon}{3} \leq \epsilon_k \leq \epsilon, k = 1, \cdots, m, \tag{22}$$

and define

$$P_k = \sum_{i=1}^{k} p_i, P_0 = 0. \tag{23}$$

Finally, for x, a q-bit fraction, let $e(x)$ be an approximation to 2^x such that

$$e(x) = 2^{x + \delta_x}, 0 \leq \delta_x \leq \frac{\epsilon}{2}. \tag{24}$$

Write p_i, P_k, and $e(x)$ with r fractional bits. Because of (21) and (22), $1 \geq P_m \geq m 2^{-r}$, so that $r \geq \log m$.

The encoding function C is defined as follows:

$$C(sa_k) = C(s) + P_{k-1} \times \Phi(sa_k), C(\lambda) = 0,$$

$$L(sa_k) = L(s) + l_k = y(sa_k) + x(sa_k),$$

$$L(\lambda) = 2r,$$

$$\Phi(sa_k) = 2^{y(sa_k)} \times e[x(sa_k)], \tag{25}$$

where $y(sa_k) = \lfloor L(sa_k) \rfloor$ and $x(sa_k)$ is the fractional part of $L(sa_k)$.

Let $u(t)$ be the largest number in $\{1, \cdots, m\}$ such that

$$P_{u(t)-1} \leq t \text{ for } t \leq 1. \tag{26}$$

Further, for the decoding we need an upper bound for a truncation of $C(s)$. We pick this as follows:

$$\overline{C}(s) = 2^{\alpha(s)}[\beta(s) + 2^{-\gamma+1}], \tag{27}$$

where $\alpha(s) = |C(s)| + 1$ and $\beta(s)$, $1 \leq \beta(s) < 2$, is defined by the γ left-most bits of $C(s)$. Put

$$\gamma = \lceil 5.6 - \log \epsilon \rceil. \tag{28}$$

The decoding function D recovers s from $\langle C(s), L(s) \rangle$ as follows:

$$s \quad = s' a_{u[t(s)]},$$

$$t(s) \quad = \overline{C}(s)/\Phi(s),$$

$$C(s') = C(s) - P_{u[t(s)]-1} \times \Phi(s),$$

$$L(s') = L(s) - l_{u[t(s)]}. \tag{29}$$

Theorem 3 If for $\epsilon > 0$,

$$\sum_{i=1}^{m} 2^{-l_i} \leq 2^{-\epsilon}, \tag{30}$$

then for every s, $D\langle C(s), L(s) \rangle = s$.

Proof We prove first that

$$C(s) \leq \Phi(s) \tag{31}$$

for every s. This inequality holds for $s = \lambda$ by (25). Arguing by induction, suppose that it holds for all strings s of length no more than n. By (25) and (31),

$$C(sa_k) \leq \Phi(s) + P_{k-1} \times \Phi(sa_k). \tag{32}$$

By (24) and (25)

$$2^{l_k - \epsilon/2}\Phi(s) \leq \Phi(sa_k) \leq 2^{l_k+\epsilon/2}\Phi(s), \tag{33}$$

which with (32) leads to

$$C(sa_k) \leq (2^{-l_k+\epsilon/2} + P_{k-1})\ \Phi(sa_k).$$

This with (22) and (23) gives

$$C(sa_k) \leq P_k\ \Phi(sa_k),$$

and because by (22)

$$P_k \leq 2^{\epsilon} \sum_{i=1}^{k} 2^{-l_i}$$

the desired inequality,

$$C(sa_k) \leq \Phi(sa_k),$$

follows with (30). The induction and the proof of (31) is complete.

When we apply the decoding algorithm to the pair $\langle C(sa_k), L(sa_k) \rangle$, we find that $u[t(sa_k)] = k$ if and only if

$$P_{k-1} \leq t(sa_k) < P_k. \tag{34}$$

If we put $\overline{C}(s) = C(s) + \delta(s)$, where

$$0 < \delta(s) \leq 2^{-\gamma+1}\ C(s) \leq 2^{-\gamma+1}\ \Phi(s), \tag{35}$$

the inequalities (34) become

$$P_{k-1} \leq \frac{C(sa_k)}{\Phi(sa_k)} + \frac{\delta(sa_k)}{\Phi(sa_k)} < P_k.$$

The former inequality holds by (25) and the fact that $\delta(sa_k) \geq 0$. The second inequality holds with the first equation of (25), and (23) translates into

$$C(s) + \delta(sa_k) < p_k\ \Phi(sa_k). \tag{36}$$

By (31), (33) and (35),

$$\begin{aligned} C(s) + \delta(sa_k) &\leq 2^{-l_k+\epsilon/2}(1 + 2^{-\gamma+1})\ \Phi(sa_k) \\ &= 2^{-l_k+\epsilon/2}(1 + 2^{-\gamma+1})\ 2^{l_k-\epsilon_k} \times p_k\ \Phi(sa_k) \\ &= (2^{(\epsilon/2)-\epsilon_k} + 2^{(\epsilon/2)-\gamma+1-\epsilon_k})\ p_k \Phi\ (sa_k). \end{aligned}$$

By (22), $\epsilon_k - \epsilon/2 \geq \epsilon/6$. Thus by a well-known exponential inequality we have

$$2^{-(\epsilon_k-\epsilon/2)} \leq 2^{-\epsilon/6} < 1 - \frac{\epsilon}{12 \log e}.$$

The coefficient of $p_k\ \Phi\ (sa_k)$ is then smaller than one for the given value (28) for γ, and (36) holds. The proof is complete.

From (31) and (25) we immediately obtain

$$\frac{1}{n} \log C(s) < \sum_i \frac{n_i}{n} l_i + \frac{2r+1}{n}.$$

Inequality (30) holds if we put

$$l_i = \log\frac{n}{n_i} + \delta_i,\ \epsilon \leq \delta_i \leq \epsilon + 2^{-q},$$

in which case

$$\frac{1}{n} \log C(s) < H\left(\frac{n_1}{n}, \cdots, \frac{n_m}{n}\right) + \epsilon + 2^{-q} + O\left(\frac{r}{n}\right),$$

where

$$H(p_1, \cdots, p_m) = \sum_i p_i \log p_i^{-1}.$$

By (33) and the fact that $P_{k-1} > 2^{-l_1+\epsilon/2}$ for $k > 1$,

$$P_{k-1}\ \Phi(sa_k) \geq 2^{l_k - l_1}\ \Phi(s).$$

It therefore follows that in the first equation of (25) no more than $2r + 1 - l_k + l_1$ left-most bits of $C(s)$ need by changed in order to get $C(sa_k)$.

The crux of the decoding process is to calculate $u(t)$ by (26) for the r-bit fractions t. This can either be tabulated or calculated from a suitable approximation to

$$P_k = \sum_{i=1}^{k} 2^{-l_i+\epsilon_i}.$$

Further, the products $P_{k-1} \times e[x(sa_k)]$ appearing in (25) can be tabulated, which is practicable for small or moderate sized alphabets. Then only two arithmetic operations per symbol are needed in the coding operations. The general case does not reduce to the binary case when $m = 2$, because the factor $P_1 = p$, approximated as 2^{-l_i}, can be absorbed by $\phi(s1)$, which is cheaper.

Finally, if for large alphabets the symbols are ordered such that the numbers l_i form an increasing sequence, the function $k \rightarrow P_k$ is concave. Then both are easy to approximate by an analytic function, and the overhead storage consists mainly of the table of symbols. The coding operations then require about the time for the table lookups $k \leftrightarrow a_k$. In addition, the l_i may be chosen, except for small values for i, in a "universal" way; e.g. as

$$l_i \cong \log i + (\log i)^{\frac{1}{2}},$$

and still have a near-entropy compression. (For universal coding, see [6]). This means that there is no need to collect elaborate statistics about the symbol probabilities. We leave the details to another paper.

References

1. D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE* **40**, 1098 (1952).
2. N. Abramson, *Information Theory and Coding*, McGraw-Hill Book Co., Inc., New York, 1963.
3. R. M. Fano, "On the Number of Bits Required to Implement an Associative Memory," *Memorandum 61*, Computer Structures Group, Project MAC, Massachusetts, 1972.
4. J. P. M. Schalkwijk, "An Algorithm for Source Coding," *IEEE Trans. Information Theory* **IT-18**, 395 (1972).
5. T. M. Cover, "Enumerative Source Encoding," *IEEE Trans. Information Theory* **IT-19**, 73 (1973).
6. P. Elias, "Universal Codeword Sets and Representations of the Integers," *IEEE Trans. Information Theory* **IT-21**, 194 (March 1975).

Received May 16, 1975; revised October 15, 1975

The author is located at the IBM Research Laboratory, Monterey and Cottle Roads, San Jose, California 95193.