Variable-Length Binary Encodings

By E. N. GILBERT and E. F. MOORE

(Manuscript received September 9, 1958)

This paper gives a theoretical treatment of several properties which describe certain variable-length binary encodings of the sort which could be used for the storage or transmission of information. Some of these, such as the prefix and finite delay properties, deal with the time delay with which circuits can be built to decipher the encodings. The self-synchronizing property deals with the ability of the deciphering circuits to get in phase automatically with the enciphering circuits. Exhaustive encodings have the property that all possible sequences of binary digits can occur as messages. Alphabetical-order encodings are those for which the alphabetical order of the letters is preserved as the numerical order of the binary codes, and would be of possible value for sorting of data or consultation of files or dictionaries. Various theorems are proved about the relationships between these properties, and also about their relationship to the average number of binary digits used to encode each letter of the original message.

I. INTRODUCTION

Table I gives three different encodings for representing the letters of the alphabet and the space symbol in binary form. These encodings have several special properties which are of some interest. First, each is a variable-length encoding; that is, the code for each letter is a sequence of binary digits, but the codes assigned to different letters are not all required to consist of the same number of binary digits. The first two of these encodings have the prefix property; that is, no one of the codes is a prefix of any other code of the same encoding. This property makes it easy to decipher a message, since it is only necessary to look at enough binary digits of the message until it agrees with one of the codes if it is desired to find the first letter of the deciphered message.

The first of these encodings, called the Huffman encoding, is constructed by the method given by Huffman,[1] and has the property of being a minimum-redundancy encoding; that is, among all variable-length binary encodings having the prefix property, this is an encoding

TABLE I

Letter	Probability	Huffman Code	Alphabetical Code	Special Code
Space	0.1859	000	00	00
A	0.0642	0100	0100	0100
B	0.0127	011111	010100	010100
C	0.0218	11111	010101	010101
D	0.0317	01011	01011	01011
E	0.1031	101	0110	0110
F	0.0208	001100	011100	011100
G	0.0152	011101	011101	011101
H	0.0467	1110	01111	01111
I	0.0575	1000	1000	1000
J	0.0008	0111001110	1001000	10001111111
K	0.0049	01110010	1001001	100100
L	0.0321	01010	100101	100101
M	0.0198	001101	10011	10011
N	0.0574	1001	1010	1010
O	0.0632	0110	1011	1011
P	0.0152	011110	110000	11000
Q	0.0008	0111001101	110001	110001111111
R	0.0484	1101	11001	11001
S	0.0514	1100	1101	1101
T	0.0796	0010	1110	1110
U	0.0228	11110	111100	111100
V	0.0083	0111000	111101	111101
W	0.0175	001110	111110	111110
X	0.0013	0111001100	1111110	1111101111111
Y	0.0164	001111	11111110	1111110
Z	0.0005	0111001111	11111111	11111101111111
Cost		4.1195	4.1978	

having the lowest possible cost (where the *cost* is defined as the average number of binary digits used per letter of the original message, assuming that the message is made up of letters independently chosen, each with the probability given).

The second of these encodings, called the alphabetical encoding, has the property that the alphabetical order of the letters corresponds to the numerical binary order of the codes. Among all such alphabetical-order-preserving binary encodings that are of variable length and have the prefix property, the one given has been constructed to have the lowest possible cost. It can be seen that the cost 4.1978 of the alphabetical encoding is quite close to the cost 4.1195 of the Huffman encoding, as compared to the cost 5 of the more conventional fixed-length encoding for the same alphabet, so that the alphabetical restriction adds surprisingly little expense to a variable-length encoding.

Part of this paper deals with the methods of constructing such best alphabetical encodings, and gives some theorems concerning their cost and their structure. However, this paper also includes theoretical results

about various properties of variable-length binary encodings in general. The cost, the prefix property and unique decipherability have already been mentioned. The exhaustive property (roughly speaking, this permits all infinite binary sequences to occur as encoded messages) is also shown to be relevant, as is the finite delay property, which has to do with the amount of delay which must take place between receiving and deciphering the enciphered message. Various theorems are proved concerning the relationships of these properties to each other and to other properties. Some of these properties have also been considered by other authors.[1,2,3,4,5,6]

One property of special interest is the ability of certain variable-length encodings (but not of fixed-length encodings) to automatically synchronize the deciphering circuit with the enciphering circuit. This self-synchronizing property, while it has been previously mentioned,[5] is a little-known property which might have practical significance in that it would permit binary deciphering machines using variable-length encodings to be built without requiring any special synchronizing circuits or synchronizing pulses, such as are needed for fixed-length encodings. Thus, there may be cases where (despite some present opinions to the contrary) variable-length encodings lend themselves to simpler instrumentation than fixed-length encodings.

Since the probabilities given in Table I are derived from one of the tables of frequencies of letters in English text,[7] the encoding given should be reasonably efficient for encoding English words or phrases. The alphabetical property, together with the prefix property, implies that two such words or phrases could be compared for alphabetical order merely by putting the two entire phrases into a simple comparison circuit of the kind which would be used to compare binary numbers. If the two phrases begin with the same sequence of letters, the corresponding parts of their enciphered form would agree, and the outcome of the binary comparison would be determined by the comparison between the two binary codes corresponding to the first pair of letters which disagree.

Placing the space symbol before the letter A of the alphabet corresponds to the usual convention governing the filing of multiple-word entries in alphabetical order, although if it were desired also to include punctuation marks or numerals in the alphabet, the conventions are not so universal, and might not be of the sort which can easily be expressed in a binary encoding.

An alphabetical encoding might be used as a means of saving memory space needed for names or other alphabetical data that are to be sorted

into alphabetical order on a data-processing machine or are to be stored in a file in alphabetical order. Similarly, it might be used for the words of a dictionary as a part of a language-translating machine, if it were desired to preserve the conventional alphabetical order of dictionaries. In addition to possible savings of memory space, it might be used to find entries in such a dictionary more quickly. Since the low redundancy of this encoding causes the digits 0 and 1 to be used with more nearly equal frequency and more nearly independently than in a fixed-length encoding, the binary numerical value associated with each word would increase more nearly as a linear function of distance progressed through the dictionary; hence, instead of searching for a given word by the method of successively halving the interval in which it is known to lie, linear interpolation (or some rough approximation to it which might be done by a simpler circuit) could be used to speed up convergence. However, for uses such as mentioned here, the particular alphabetical encoding given in Table I is not necessarily the optimum, since the frequencies of occurrence of letters in names or in dictionary entries are undoubtedly different than they are in connected English text. However, the methods given in this paper would enable such an encoding to be obtained for any given probability distribution.

II. TERMINOLOGY

We will use the word *letter* to refer to any symbol of some designated list, including even the space symbol of Table I. By an *alphabet* we will mean a set of letters. We will usually require each member of an alphabet to have associated with it a probability of occurrence, and we will also usually require that some linear ordering relationship (which we will call *alphabetical order*) be defined for the letters of this alphabet. So that we may call any subset of the letters of an alphabet a *subalphabet*, and may keep the same ordering and the same probabilities, we will require only that the sum of the probabilities be less than or equal to one. All of the alphabets considered in this paper have only a finite number n of letters, but it might be advisable to allow countably infinite alphabets in certain further theoretical extensions of this subject.

A *message* is a finite sequence of letters, or an infinite sequence $L_1L_2L_3 \cdots$ which extends infinitely only into the future, not into the past. We will consider a *source* which generates messages in which successive letters occur independently and with the given probabilities. However, in case the sum of the probabilities is less than one, we may imagine that the probabilities are proportionately increased just enough that their sum becomes one, so that the associated source is more realistic.

We distinguish between code and encoding, both of which are often called codes by other writers. A *code* is a finite sequence of binary digits. An *encoding* is a way of associating (or more formally, a function C which associates) a code C_i with each letter L_i of an alphabet.

The operation of *enciphering* (elsewhere often called encoding) constructs a sequence of binary digits which is made up of the code for the first letter of the message, followed immediately by the code for the second letter of the message, etc. Any message then produces a sequence of binary digits called the *enciphered message*. Any machine or circuit which does the operation of enciphering is called an *enciphering machine* or an *enciphering circuit*. The enciphered message of a finite message is obviously always finite.

An encoding will be said to be *uniquely decipherable* if, for each finite enciphered message, there exists exactly one original message which could have produced it. If an encoding is uniquely decipherable, then there is obviously a procedure for deciphering any finite enciphered message (by enumeration, for instance), and any machine or circuit capable of doing this will be called a *deciphering machine* or a *deciphering circuit*.

Following Huffman,[1] we define a *prefix* of any sequence Φ of binary digits to be any finite sequence which is either Φ itself or is obtainable by deleting all of the digits after a given point of Φ. For example, the prefixes of 10110 are 10110, 1011, 101, 10, 1, and the *null sequence*, which has no digits. We will say that an encoding C has the *prefix property* if no code of C is a prefix of any other code of C.

By a *presumed message* we will mean a finite or infinite sequence Φ of binary digits such that every prefix of Φ is a prefix of the enciphered form of some message. Then, at any given time while a presumed message is being sent into a deciphering machine, it is indistinguishable from a message, so it makes sense to allow presumed messages as well as messages to be the class of sequences which can be sent into a deciphering machine.

III. THE ENCODING THEOREM FOR ALPHABETICAL ENCODINGS

Consider a discrete source S which uses the alphabet: space, A, B, $\cdots$, Z (any other linearly ordered alphabet will also serve). An encoding of blocks of N letters into binary sequences will be called an *alphabetical encoding* if it is uniquely decipherable and the codes for the blocks in alphabetical (dictionary) order are themselves in numerical order. Here the codes are imagined to be prefixed by binary points to convert them into numbers in binary form. The alphabetical encoding of Table I is a

case with $N = 1$. It is a natural question to ask if a restriction to alphabetical encodings may not be severe for some sources S. In particular, are the results of Shannon's encoding theorem (Ref. 8, Theorem 9) still obtainable with alphabetical encodings?

Shannon proved that the output of a discrete source having entropy H bits per character can be enciphered in a uniquely decipherable manner into a sequence of binary digits so that the average number of digits used per character exceeds H by an arbitrarily small amount. Shannon's construction encodes blocks of N source characters into binary sequences, using a cost (average number of binary digits per character) H_N' which satisfies

$$NG_N \leqq NH_N' \leqq NG_N + 1. \tag{1}$$

Here, NG_N is Shannon's notation for the information contained in a block of N characters produced by the source; i.e.,

$$NG_N = -\sum_i p_i \log p_i , \tag{2}$$

in which the p_i are the N-gram probabilities of the source. Then, since

$$\lim_{N\to\infty} G_N = H,$$

Shannon's theorem

$$\lim_{N\to\infty} H_N' = H \tag{3}$$

follows from (1). Since NG_N must be a lower bound on the average number of digits used to encode a block of N characters by any means whatever, (1) shows that Shannon's construction is not far from the best possible one for block encoding. We now give a similar theorem for alphabetical encoding.

Theorem 1: Let S be a source producing messages which may be ordered (alphabetically). Let G_N be computed from the N-gram probabilities p_i of S by (2). There exists a uniquely decipherable alphabetical encoding of blocks of N characters of S into sequences of binary digits for which the cost, H_N, satisfies

$$NG_N \leqq NH_N \leqq NG_N + 2. \tag{4}$$

By picking N large enough, H_N may be made arbitrarily close to the entropy H of S in bits per character.

Proof: The proof is adapted from Shannon's[8] proof of his Theorem 9. Let all possible blocks of N source characters be listed in alphabetical order, and let p_i denote the probability of the ith block in the list (recall

that Shannon lists his blocks in order of probability rather than alphabetically). Let m_i be the integer for which

$$2^{-m_i} \leqq p_i < 2^{1-m_i}.$$

Also, let numbers $A_1, A_2, A_3, \cdots$ be defined by

$$\begin{aligned} A_1 &= \frac{p_1}{2}, \\ A_2 &= p_1 + \frac{p_2}{2}, \\ &\vdots \qquad\qquad \vdots \\ A_i &= (p_1 + \cdots + p_{i-1}) + \frac{p_i}{2}. \end{aligned} \tag{5}$$

Note that

$$0 \leqq A_1 \leqq A_2 \leqq \cdots \leqq 1.$$

We now construct an alphabetical encoding. The code for the ith block will be the first $m_i + 1$ digits of the binary expansion of the number A_i. In Shannon's encoding this same block has a code formed by expanding a (different) number to m_i places. Then our scheme uses only one more digit than does Shannon's for each block, $NH_N = NH'_N + 1$, and (4) follows from (1). It remains now to show that our encoding is uniquely decipherable; i.e., that the sequence of letters generated by S may be reconstructed from the binary digits.

It suffices to prove that our construction produces a list of codes which have the prefix property. Then the enciphered message produced by each block of N letters may be deciphered as soon as all its digits have been received.

To prove that our list has the prefix property, consider any two blocks of letters, say the ith and the jth with $i < j$. By (5),

$$A_j \geqq A_i + \frac{p_j}{2} + \frac{p_i}{2},$$

and

$$A_j \geqq A_i + 2^{-1-m_j} + 2^{-1-m_i}. \tag{6}$$

If $p_i \leqq p_j$, then $m_i \geqq m_j$; but, by (6), the jth code cannot be identically the same as the first $1 + m_j$ places of the ith code. Similarly, if $p_i \geqq p_j$ the ith code cannot be a prefix of the jth code. Thus, the prefix property, and the theorem, follow.

Except in the case of an alphabet having only one letter, the prefix property is sufficient to insure unique decipherability, but it is not necessary. For example, the list 0, 01, 11 does not have the prefix property; still it could be used. In a received message 00001111 $\cdots$ there would be no doubt about the first three 0's, and the fourth 0 would be recognized as 01 or not according to whether an odd or even number of 1's followed it.

However, by a *best alphabetical encoding* we will mean an encoding which has the lowest cost among all alphabetical encodings which have the prefix property. This insistence upon the prefix property will make it possible for us to prove Theorems 2 through 5 and give constructive methods for finding these best alphabetical encodings.

If we use the construction just described to design an alphabetical encoding of English with $N = 1$, we obtain a cost of 5.75 digits per character. As guaranteed by the theorem, this cost is less than $G_1 + 2 = 6.08$. However, we could have done better by simply assigning a five-digit code to each letter. The encoding can be much improved by

TABLE II

Letter	Code	Shortened Code
Space	0001	000
A	00110	001
B	01000001	010000
C	0100011	010001

deleting some digits which are obviously not needed. For example, the first few codes are those listed in Table II. Clearly the code 00110 for A is too long. As soon as the prefix 001 is received, A is the only possibility. The final digits 10 may be deleted. Similarly, the other codes may be shortened, as indicated in Table II, until no code can lose a final digit without becoming a prefix for some other code. The cost is thereby reduced to 4.44 digits per character.

A different encoding is obtainable using the same sort of construction but with

$$A_i = \sum_{j=1}^{i-1} 2^{-m_j} + 2^{-m_i-1}.$$

The same proof can be used, since (6) still holds. Since the code lengths are again the numbers $m_i + 1$, the new encoding will have the same cost. The numbers A_j can now be computed with ease directly in the binary system, and much of the arithmetic needed for the first construction may be avoided. However, the kind of shortening used in Table II does

not work as well with the new encoding. All codes (as numbers) are now less than

$$\sum_i 2^{-m_i}.$$

This number need not be near 1 (typically it is about $\frac{3}{4}$). The codes are then cramped together in a range smaller than (0,1) and cannot be shortened as much. For the case of the English source with $N = 1$, the new encoding can only be shortened to cost 5.02 digits per letter.

IV. ENCODING TRICKS

The simple construction just given does not produce the best encoding, i.e., the one with least cost. The best encoding can always be found by a systematic, although long, calculation which is described in the next section. Here we list a few tricks whereby the problem of finding the best encoding may be simplified and, in some cases, solved.

We will describe these results in terms of encoding single letters into binary form; however, it is to be understood that blocks of N letters may always be considered the single letters of a larger alphabet. By a *prefix set* of an encoding we will mean the set of all letters which have codes beginning with a given prefix. For example, in the Huffman encoding of Table I the prefix 011 has the prefix set consisting of letters B, G, J, K, P, Q, V, X and Z. In an alphabetical encoding every prefix set must consist of all letters lying between some two fixed letters in the alphabet.

The tricks to be described enable one to prove that certain collections of letters must be prefix sets in any best alphabetical encoding. Whenever a prefix set is known the encoding problem can then be reduced as follows to one for a smaller alphabet.

Theorem 2: In a best alphabetical encoding let S be a prefix set for a prefix π. Construct a shorter alphabet by replacing the letters of S by a single new letter, L^1, occupying their place in alphabetical order and having as its probability the sum of their probabilities. A best encoding of the new alphabet gives L^1 the code π and gives every other letter its old code.

Proof: Let $C(L)$ denote the code for letter L in the original best encoding. Suppose, contrary to the theorem, that the new problem had a better solution in which L, L^1 had codes $C^1(L)$ and $C^1(L^1)$. One would then obtain a better solution of the original problem by encoding L into $C^1(L)$. The code for a letter M in the prefix class would be $C(M)$ with the prefix π changed to $C^1(L^1)$.

Huffman's encoding scheme uses a result similar to Theorem II for

nonalphabetical encodings. The two letters of lowest probability must form a prefix set, and his result is used again and again, until there are only two letters left and the problem is solved. When the encoding must be alphabetical one cannot always find a prefix set easily. Some results in this direction are given by the following theorems. The symbols L_1, L_2, $\cdots$ are used to represent the letters of the alphabet in order; p_1, p_2, $\cdots$ will be their probabilities; $C(L_1)$, $C(L_2)$, $\cdots$ will be their codes in the encoding C and N_1, N_2, $\cdots$ will be the numbers of binary digits in their codes. Also, if Φ is any code or any prefix, $N(\Phi)$ will be used to represent the number of binary digits in Φ.

An encoding will be said to be *exhaustive* if it encodes an alphabet of two or more letters in a uniquely decipherable manner and, for every infinite sequence $x = x_1x_2x_3 \cdots$ of binary digits, there is some message which can be enciphered as x; or if it encodes an alphabet of one letter by using the null sequence.

Theorem 3: Every best alphabetical encoding is exhaustive.

Proof: Consider an encoding of an alphabet having two or more letters which is alphabetical and has the prefix property, but is not exhaustive. It will be shown that it is not a best encoding. Let x be an infinite sequence of binary digits such that no message can be encoded as x. If any code of the encoding is a prefix of x, remove it from x, and, after a finite number of repetitions of this process, an x will be obtained which has no one of the codes for a prefix. Let Φ be the greatest prefix of x which is also a prefix of any one of the codes. Let C_i be some code of which Φ is a prefix. We will use $\Phi 0$ to represent the sequence Φ followed by 0. Then either $\Phi 0$ is a prefix of C_i and $\Phi 1$ is a prefix of x, and $\Phi 1$ is not a prefix of any code of this encoding; or else $\Phi 1$ is a prefix of C_i and $\Phi 0$ is a prefix of x, and $\Phi 0$ is not a prefix of any code of this encoding. Without loss of generality, we assume the second one of these alternatives. Then consider the new encoding which agrees with the old one for all codes not having Φ as a prefix, but which has a code $\Phi\theta$ in place of each code of the form $\Phi 1\theta$. The new encoding has a lower cost than the old one, is still alphabetical and still has the prefix property. Hence the original encoding was not a best alphabetical encoding.

Lemma 1: Let π be a prefix. In a best alphabetical encoding, if there is a code with prefix $\pi 0$ there is one with prefix $\pi 1$. Conversely, if there is a code with prefix $\pi 1$, there is one with prefix $\pi 0$.

Proof: If $\pi 0$ is a prefix, then by Theorem 3 the sequence $\pi 111 \cdots$ must have some code C_i as a prefix. But by the prefix property, C_i cannot be a prefix of $\pi 0$; hence, C_i has prefix $\pi 1$. The converse is proved similarly.

Lemma 2: Let L_a be the letter of lowest probability. In a best alphabetical encoding, L_a, together with one of L_{a+1} or L_{a-1} must form a prefix set.

Proof: Suppose $C(L_a)$ ends in 0, say $C(L_a) = \pi 0$, where π stands for some prefix. By Lemma 1, $\pi 1$ is a prefix of $C(L_{a+1})$. If $C(L_{a+1}) = \pi 1$, we have the desired result. If not, $\pi 10$ must be a prefix of $C(L_{a+1})$. By Lemma 1 there exist codes with prefix $\pi 11$. A better encoding (and hence a contradiction) may be had by the following changes: Lengthen $C(L_a)$ from $\pi 0$ to $\pi 00$. Change all codes of the form $\pi 10\psi$ to $\pi 01\psi$. Shorten all codes of the form $\pi 11\psi$ to $\pi 1\psi$. Since the last change applies to at least one letter (of higher probability than L_a), there is a net decrease in cost.

The proof in the other case [$C(L_a)$ ending in 1] is similar. If, as is the case of the probabilities of Table I, the least probable letter is at the end of the alphabet, then this letter has only one neighboring letter and must form a prefix set with it. Thus, as a first step in Table I, we can write

$$C(\mathrm{Y}) = \pi(\mathrm{Y},\mathrm{Z})0,$$
$$C(\mathrm{Z}) = \pi(\mathrm{Y},\mathrm{Z})1,$$

where $\pi(\mathrm{Y},\mathrm{Z})$ is some unknown prefix. Then, using Theorem 2, the problem is reduced to an encoding for a 26-letter alphabet in which Y and Z have been replaced by a single letter $L(\mathrm{Y},\mathrm{Z})$ of probability 0.0169. When this new problem is solved, $\pi(\mathrm{Y},\mathrm{Z})$ will be found as the code for $L(\mathrm{Y},\mathrm{Z})$. The new least probable letter is J or Q, both with the same probability 0.0008; J, for example, can be in a prefix set with either I or K, but Lemma 2 gives no clue for deciding which one. One might hope that one can always pick the less probable neighbor, K in this case. However, it is easy to find counter-examples which disprove this conjecture. A weaker, but true, theorem is the following one.

Theorem 4: Let L_a be the letter of lowest probability. Suppose that

$$p_{a+1} > p_a + p_{a-1}. \tag{7}$$

Then L_a and L_{a-1} must form a prefix set in any best alphabetical encoding. Similarly, if $p_{a-1} > p_a + p_{a+1}$, L_a and L_{a+1} must form a prefix set.

Proof: Suppose (7) holds but that L_a and L_{a-1} do not form a prefix set. Then, by Lemma 2, L_a and L_{a+1} form a prefix set. The codes for L_a and L_{a+1} must be of the form

$$C(L_a) = \pi 0,$$
$$C(L_{a+1}) = \pi 1$$

for some prefix π. The code $C(L_{a-1})$ must end in 1, say $C(L_{a-1}) = \rho 1$.

For, if $C(L_{a-1})$ were $\rho 0$, Lemma 1 would show that some code has prefix $\rho 1$ and hence must stand for a letter between L_{a-1} and L_a in the alphabetical order, an impossibility. Lemma 1 now shows that some other letters have prefix $\rho 0$.

We consider two cases determined by the numbers $N(\pi)$ and $N(\rho)$ of digits in π and ρ:

Case 1 — $N(\pi) < N(\rho)$. An improved encoding can be made by changing $C(L_a)$ from $\pi 0$ to $\pi 01$, $C(L_{a-1})$ from $\rho 1$ to $\pi 00$ and all codes of the form $\rho 0\psi$ to $\rho\psi$. The last change, a shortening, affects some codes and so offsets the lengthening of the least probable code.

Case 2 — $N(\rho) \leqq N(\pi)$. An improvement can be made by shortening $C(L_{a+1})$ from $\pi 1$ to π while changing $C(L_a)$ from $\pi 0$ to $\rho 11$ and $C(L_{a-1})$ from $\rho 1$ to $\rho 10$. That there is a net decrease in cost follows from (7).

The other half of the theorem is proved in a similar way.

Applying Theorem 4 to our reduced problem of Table I, we obtain further reductions, producing new letters L(J,K) and L(P,Q) with probabilities 0.0057 and 0.0160. Now the lowest-probability letter has become X, and we need another kind of theorem.

Theorem 5: If L_i and L_j $(i < j)$ are two letters both of probability exceeding $p_{i+1} + p_{i+2} + \ldots + p_{j-1}$, then the intervening letters $L_{i+1}, L_{i+2}, \ldots, L_{j-1}$ form a prefix set in any best alphabetical encoding.

Proof: Let π denote the greatest common prefix of $C(L_i)$ and $C(L_j)$, i.e., a prefix such that $\pi 0$ is a prefix of $C(L_i)$ while $\pi 1$ is a prefix of $C(L_j)$. The intervening letters have either $\pi 0$ or $\pi 1$ as prefixes. Supposing that there are some intervening letters with prefix $\pi 0$, we assert that *the intervening letters with prefix $\pi 0$ form a prefix set.* To prove this assertion, let the intervening letters with prefix $\pi 0$ be $L_{i+1}, \ldots L_c$, where $C(L_{c+1})$ has prefix $\pi 1$. Let $\pi 0\rho$ denote the greatest common prefix of $C(L_i)$ and $C(L_c)$. Then $C(L_c)$ must have prefix $\pi 0\rho 1$; otherwise, by Lemma 1, L_{c+1} would have prefix $\pi 0\rho$, and hence $\pi 0$. Also, $C(L_i)$ has prefix $\pi 0\rho 0$; otherwise, $\pi 0\rho 1$ would be a greater common prefix than $\pi 0\rho$. The assertion requires only that we prove that $C(L_{i+1})$ has prefix $\pi 0\rho 1$, for then the letters in question and no others have this prefix. If, on the contrary, $C(L_{i+1})$ has prefix $\pi 0\rho 0$, find the greatest common prefix $\pi 0\rho 0\sigma$ such that $\pi 0\rho 0\sigma 0$ is a prefix of $C(L_i)$ and $\pi 0\rho 0\sigma 1$ is a prefix of $C(L_{i+1})$. Now shorten all codes of the form $\pi 0\rho 0\sigma 0\psi$ to $\pi 0\rho 0\sigma\psi$ and lengthen all other codes $\pi 0\rho\psi$ to $\pi 0\rho 1\psi$. The shortened codes include the one for L_i, which has more probability than the total probability of all the lengthened codes. The assertion is now proved, and likewise *intervening letters with prefix $\pi 1$ form a prefix set.*

By our two assertions, each of $C(L_{i+1}), \ldots, C(L_{j-1})$ has one of two

prefixes, which we may call $\pi 0\rho 1$ and $\pi 1\tau 0$, while $\pi 0\rho 0$ is a prefix of $C(L_i)$ and $\pi 1\tau 1$ is a prefix of $C(L_j)$. Again, one proves the theorem by making changes which put the intervening letters into a single prefix set. There are two cases:

Case 1 — $N(\pi 0\rho) \leqq N(\pi 1\tau)$. Lengthen codes $\pi 0\rho 1\psi$ to $\pi 0\rho 10\psi$. Change codes $\pi 1\tau 0\psi$ to $\pi 0\rho 11\psi$. Shorten all codes $\pi 1\tau 1\psi$ to $\pi 1\tau\psi$. The intervening letters now form a prefix set with prefix $\pi 0\rho 1$ and the new encoding has smaller cost.

Case 2 — $N(\pi 1\tau) \leqq N(\pi 0\rho)$. By changes similar to those of Case 1, one may reduce the cost by making the intervening letters into a prefix set with prefix $\pi 1\tau 0$.

Applying Theorem 5 to Table I, we now recognize new prefix sets and reduce the problem by introducing new letters L(F,G) and L(U,V, W,X,Y,Z) of probabilities 0.0360 and 0.0668. Now L(J,K) becomes the least probable letter, Theorem 4 applies, and we form a new letter L(J,K,L) of probability 0.0378. Next, Theorem 4 applies to letter B, and we form a new letter L(B,C) of probability 0.0345. Again we are at an impasse.

Theorem 6: If $p_1 < p_3$, then L_1 and L_2 form a prefix set in any best alphabetical encoding. Similarly, if L_n is the last letter of the alphabet, L_{n-1} and L_n must form a prefix set if $p_n < p_{n-2}$.

Proof: If $p_1 < p_3$ and L_1 and L_2 are not a prefix set, then $C(L_1)$, $C(L_2)$ and $C(L_3)$ may be shown to have the forms $\pi 0$, $\pi 1\rho 0$ and $\pi 1\rho 1\psi$. Then one could improve the encoding by changing $C(L_1)$ to $\pi 00$, $C(L_2)$ to $\pi 01$ and all codes $\pi 1\rho 1\psi$ to $\pi 1\rho\psi$.

This theorem provides no further reduction of our example. Note, however, that it might have been applied following the creation of L(Y,Z) to prove that X,Y,Z, forms a prefix set. This information is helpful when we must add the final digits to the prefix π(U,V, . . . , Z) to form the codes for U, . . . , Z. Using Huffman's encoding method, we find, disregarding questions of alphabetical order, the best way of encoding four letters which have probabilities in the same ratio as our letters U,V,W and L(X,Y,Z). The solution gives each letter two digits. Then, an equally good alphabetical encoding gives these letters the code 00, 01, 10, 11. We now know parts of the codes sought, as summarized in Table III. The unknown prefixes π(B,C), . . . are to be determined by finding a best alphabetical encoding of the 17-letter alphabet listed in Table IV.

Again we might try a Huffman encoding for Table IV. However, we note in advance that M and L(P,Q) are much less probable than their neighbors. Then a Huffman encoding will give these letters such long

TABLE III

Letter	Code
B	π(B,C)0
C	π(B,C)1
F	π(F,G)0
G	π(F,G)1
J	π(J,K,L)00
K	π(J,K,L)01
L	π(J,K,L)1
P	π(P,Q)0
Q	π(P,Q)1
U	π(U, $\cdots$, Z)00
V	π(U, $\cdots$, Z)01
W	π(U, $\cdots$, Z)10
X	π(U, $\cdots$, Z)110
Y	π(U, $\cdots$, Z)1110
Z	π(U, $\cdots$, Z)1111

codes that there will be no alphabetical encoding which uses the same length codes for every letter. To circumvent this difficulty we use Lemma 2, first on L(P,Q) and next on M, and conclude that L(P,Q) must form a prefix set with O or R and M must form a prefix set with L(J,K,L) or N. There are then four new alphabets to consider, and we have constructed Huffman encodings for each one. The one with smallest cost is the one in which J,K,L,M and P,Q,R were made into new letters. The numbers of digits for the letters in Table IV which this Huffman encoding required are listed. We next look for an alphabetical encoding in which the same numbers of digits is used. Such an encoding actually

TABLE IV

Letter	Probability	Number of Digits
Space	0.1859	2
A	0.0642	4
L(B,C)	0.0345	5
D	0.0317	5
E	0.1031	4
L(F,G)	0.0360	5
H	0.0467	5
I	0.0575	4
L(J,K,L)	0.0378	5
M	0.0198	5
N	0.0574	4
O	0.0632	4
L(P,Q)	0.0160	5
R	0.0484	5
S	0.0514	4
T	0.0796	4
L(U, $\cdots$,Z)	0.0668	4

exists, and so we obtain the best alphabetical encoding shown in Table I. It must be admitted that we were somewhat lucky to be able to reduce the problem to one in which one of the best possible encodings, disregarding alphabetical order, includes an alphabetical encoding. Undoubtedly, minor changes in the probabilities in Table I might make the problem much harder. In the next section we give an encoding method which will apply in all cases.

V. THE GENERAL ALPHABETIZING ALGORITHM

The method which will be used in general builds up the best alphabetical encoding for the entire alphabet by first making best alphabetical encodings for certain subalphabets. In particular, the subalphabets which will be considered will be only those which might form a prefix set in some alphabetical binary encoding of the whole alphabet. Since only those sets of letters consisting exactly of all those letters which lie between some pair of letters can serve as a prefix set, we will call such a set an *allowable* subalphabet.

We will denote the allowable subalphabet consisting of all of those letters which follow L_i in the alphabet (including L_i itself) and which precede L_j (again including L_j itself) by (L_i , L_j). When referring to the ordinary English alphabet of Table I we will use the symbol # for the space symbol. Thus, (#,B) will be the subalphabet containing the three symbols space, A and B, and (A,A) will be used to denote the subalphabet containing only the letter A.

If it were desired to find an optimum encoding satisfying certain kinds of restrictions other than the alphabetical one, different allowable subalphabets could be used, with the rest of the algorithm remaining analogous. This method of building up an encoding by combining encodings for subalphabets is analogous to the method used by Huffman,[1] except that he was able to organize his algorithms such that no subalphabets were used except those which actually occurred as prefix sets in his final encoding. However, we consider all allowable subalphabets, including some which are not actually used as part of the final encoding.

The term cost of an encoding has been used to refer to the average number of binary digits per letter of transmitted message, that is, $\sum_i p_i N_i$. Since, in the algorithm to be described, we will be constructing an encoding for each allowable subalphabet, we will also use the corresponding sum for each subalphabet. But, since the probabilities p_i do not even add up to 1 for proper subalphabets, the sum $\sum_i p_i N_i$ does not correspond exactly to a cost of transmitting messages, and so the corresponding sum will be called a *partial cost*.

The algorithm to be described takes place in n stages, where n is the number of letters in the alphabet. At the kth stage, the best alphabetical binary encoding for each k-letter allowable subalphabet will be constructed and its partial cost will be computed. For $k = 1$, each subalphabet of the form (L_i, L_i) will be encoded by the trivial encoding which encodes L_i with the null sequence; it has cost 0, since the number of digits in the null sequence is zero. For $k = 2$, each subalphabet of the form (L_i, L_{i+1}) will be encoded by letting the code for L_i be 0 and the code for L_{i+1} be 1. The partial cost of this encoding is $p_i + p_{i+1}$. In general, the kth stage of the algorithm, in which it is desired to find the best alphabetical binary encoding for each subalphabet of the form (L_i, L_{i+k-1}) and its partial cost, proceeds by making use of the codes and the partial costs computed in the previous stages.

For each j between $i + 1$ and $i + k - 1$, we can define a binary alphabetical encoding as follows: Let $C_i, C_{i+1}, \ldots C_{j-1}$ be the codes for $L_i, L_{i+1}, \ldots L_{j-1}$ given by the (previously constructed) best alphabetical encoding for (L_i, L_{j-1}), and let $C'_j, C'_{j+1}, \ldots, C'_{i+k-1}$ be the codes for $L_j, L_{j+1}, \ldots, L_{i+k-1}$ given by the (previously constructed) best alphabetical encoding for (L_j, L_{i+k-1}). Then the new encoding for $L_i, L_{i+1}, \ldots, L_{j-1}, L_j, L_{j+1}, \ldots, L_{i+k-1}$ will be $0C_i, 0C_{i+1}, \ldots, 0C_{j-1}, 1C'_j, 1C'_{j+1}, \ldots, 1C'_{i+k-1}$. Such an encoding can be defined for each j, and the encoding is exhaustive. It follows from Theorem 2 that the best encoding for this subalphabet is given by one of the $k - 1$ such encodings which can be obtained for the $k - 1$ different values of j. The partial cost of such an encoding made up out of two subencodings is the sum of the partial costs of the two subencodings plus $p_i + p_{i+1} + \ldots + p_{i+k-1}$. To perform the algorithm it will not be necessary to construct all of these encodings, but only to compute enough to decide which one of the $k - 1$ different encodings has the lowest partial cost. This is done by taking the sums of each of the $k - 1$ pairs of partial costs of subencodings and constructing the best encoding only.

After the kth stage of this algorithm has been completed for $k = 1, 2, \ldots, n$, the final encoding obtained is the best alphabetical encoding for the entire original alphabet, and the final partial cost obtained is the cost of this best alphabetical encoding.

If the above algorithm were performed on a digital computer, the length of time required to do the calculation would be proportional to n^3. The innermost inductive loop of the computer program would perform the operation mentioned above of computing sums of pairs of partial costs, and this would be done $k - 1$ times in the process of encoding each one of the subalphabets considered in the kth stage. But,

since there are $n - (k - 1)$ different allowable subalphabets to be encoded in the kth stage, there are $(k - 1)[n - (k - 1)]$ steps to be done in the kth stage. To find the total number of operations done in all of the stages, we sum, and find that

$$\sum_{k=1}^{n} (k - 1)[n - (k - 1)] = \frac{(n^3 - n)}{6},$$

which is an identity which can be verified by mathematical induction.

VI. PROPERTIES OF EXHAUSTIVE ENCODINGS

We have already shown (Theorem 3) that every best alphabetical encoding is exhaustive. Another reason for considering exhaustive encodings to be of some general interest is given by the following theorem.

Theorem 7: The Huffman binary encoding of any alphabet is exhaustive.

Proof: We prove by induction that each of the encodings for prefix sets arrived at during the steps of the algorithm of Huffman[1] is an exhaustive encoding. If this holds for the first k encodings constructed during this algorithm, consider the prefix set L encoded at the $(k + 1)$th step. Let $x = x_1x_2x_3 \ldots$ be any infinite sequence of binary digits. It suffices to show that there is some letter whose code is a prefix of x. The set L was made by combining two previous prefix sets of letters, L' and L'', and it was encoded by prefixing the codes from their previous encodings by 0 and 1 respectively. Let L' be the set whose codes were prefixed by x_1. Then if L' is a single letter, x_1 is its code, and hence its code is a prefix of x. But if L' is a prefix set, then its previous encoding is exhaustive by inductive hypothesis, and hence there is a letter L''' whose previous code is a prefix of $x_2x_3 \ldots$. Then the new code for L''' is a prefix of x.

Several of the properties of exhaustive encodings will be considered, since both the Huffman encoding and the best alphabetical encoding are exhaustive, and it seems likely that exhaustive encodings might arise from other types of optimizing problems. For instance, the shortening procedure used in Table II was essentially a way of making the encoding more nearly exhaustive.

Lemma 3: Whenever an encoding C has the property that for any infinite sequence $x = x_1x_2x_3 \ldots$ there is a code of C which is a prefix of x, then

$$\sum_{i=1}^{n} 2^{-N_i} \geqq 1, \tag{8}$$

and equality holds if and only if C has the prefix property.

Proof: Consider the set P of all finite sequences x having length exactly k, where k is some fixed integer longer than the longest code of C. Then the property assumed in the hypothesis implies that each element of P has at least one of the codes for a prefix. But P has exactly 2^k elements, and for each code of length N_i there are 2^{k-N_i} elements of P of which it is a prefix. Hence,

$$\sum_{i=1}^{n} 2^{k-N_i} \geqq 2^k,$$

which is equivalent to (8), and equality holds if and only if no element of P has two different codes for a prefix. However, the occurrence of two different codes which are prefixes of the same sequence is exactly equivalent to having one of the two codes be a prefix of the other.

Theorem 8: Every exhaustive binary encoding has the prefix property and satisfies

$$\sum_{i=1}^{n} 2^{-N_i} = 1. \tag{9}$$

Proof: By Lemma 3 and the definition of exhaustive, (8) holds, but, by McMillan,[3] unique decipherability implies

$$\sum_{i=1}^{n} 2^{-N_i} \leqq 1. \tag{10}$$

Then we combine (8) and (10) to obtain (9). But, by Lemma 3, this implies the prefix property.

Lemma 4: For any exhaustive encoding of an alphabet, and any prefix Φ of this encoding, the new encoding of the prefix-set subalphabet which associates the new code θ with each letter whose original code was $\Phi\theta$ is an exhaustive encoding of this subalphabet.

Proof: Given any x, to find a letter whose new code is a prefix of x we consider the letter L whose original code was a prefix of Φx. Then, by the prefix property, the original code of L cannot be a prefix of Φ, and thus the original code of L is of the form $\Phi\theta$. Hence, L is in the subalphabet, its new code is θ, and θ is a prefix of x. To complete the proof that the new encoding is exhaustive, note that it has the prefix property because the original encoding does. Hence, the new encoding is either the trivial encoding (of a one-letter alphabet) or is uniquely decipherable.

Lemma 5: For any exhaustive binary encoding of an alphabet having n letters, the total number of prefixes is $2n - 1$.

Lemma 6: In any exhaustive binary encoding of an alphabet having n letters, none of the codes consist of more than $n - 1$ digits.

Each of the last two lemmas associates a number with each exhaustive encoding, and they can be proved by induction on the number of letters in the alphabet. The number associated with each exhaustive encoding is represented in terms of the number associated with each of the two encodings that are constructed as described in Lemma 4 for the subalphabet having the prefix 0 and the subalphabet having the prefix 1.

Theorem 9: The cost of the Huffman encoding of an alphabet is a continuous function of the probabilities of the letters.

Theorem 10: The cost of the best alphabetical encoding of an alphabet is a continuous function of the probabilities of the letters.

The last two theorems will be proved together, enclosing in parentheses the changes which convert the proof of Theorem 9 into a proof for Theorem 10. In fact, what will be proved are the slightly stronger theorems: For two alphabets A and A^* having the same n letters, if p_i is the probability of the ith letter of A, p_i^* is the probability of the ith letter of A^*, and if k and k^* are the costs of the Huffman encoding (best alphabetical encoding) for A and A^*, then

$$| k - k^* | \leqq (n - 1) \sum_{i=1}^{n} | p_i - p_i^* |. \tag{11}$$

If we let B be the right member of inequality (11) and let k' be the cost of using the Huffman (best alphabetical) encoding of A^* as an encoding for A, then, by Lemma 6 and the definition of cost, we can conclude that $| k' - k^* | \leqq B$ and, since from the definition of k we can conclude that $k \leqq k'$, we can combine these to obtain $k^* - k \leqq B$. By a similar argument involving the use of k'', the cost of using the Huffman (best alphabetical) encoding of A as an encoding for A^*, we obtain $k - k^* \leqq B$. Combining these, we obtain (11).

Theorem 11: The Huffman encoding for a given alphabet has a cost which is less than or equal to that of any uniquely decipherable encoding for that alphabet.

Proof: This proof is essentially that of McMillan.[3] Let us consider any uniquely decipherable encoding C. We will construct a new encoding C' which has the same cost as C, and which has the prefix property. However, by its method of contruction, the Huffman encoding has a cost which is less than or equal to that of any encoding having the prefix property, completing the proof of the theorem. Let N_i be the number of digits in the code which C associates with the ith letter of the alphabet. Let the letters of the alphabet be renumbered in such a way that $N_i \leqq N_{i+1}$. Then, as in the encoding theorem (Theorem 1 of this paper, or Theorem 9 of Shannon,[8] we let

$$A_i = \sum_{j=1}^{i-1} 2^{-N_j},$$

and we define C' to be the encoding which associates with the ith letter the code C_i' obtained by truncating A_i after N_i digits. Then it follows that the digits truncated were 0's, and hence that each C_i' agrees numerically with the corresponding A_i. By (10), each of the A_i is less than 1. To show that C' has the prefix property, we assume that C_i' is a prefix of C_j'. Then $i < j$, by the renumbering. However, $A_{i+1} = A_i + 2^{-N_i}$, and hence $A_j \geqq A_i + 2^{-N_i}$. Thus, A_j cannot agree with the first N_i places of A_i. Hence, the first N_i digits of C_j' are different from those of C_i'.

Theorem 12: If A_n is the number of exhaustive binary alphabetical encodings for an alphabet having n letters, $A_1 = A_2 = 1$, and for $n \geqq 3$ we have

$$A_n = \frac{(2n-3)!\,2}{(n-2)!\,n!}. \tag{12}$$

Theorem 13: If T_n is the total number of exhaustive binary encodings for an alphabet having n letters, $T_1 = 1$, $T_2 = 2$ and, for $n \geqq 3$, we have

$$T_n = \frac{(2n-3)!\,2}{(n-2)!}. \tag{13}$$

These theorems show how rapidly A_n and T_n increase with increasing n. Since, by Theorem 3, A_n would be the number of encodings to consider if it were desired to find the best alphabetical encoding by enumeration, Theorem 12 shows that the methods already given in this paper (even the general alphabetizing algorithm) are much faster than exhaustive enumeration. Similarly, Theorem 7 and Theorem 8 show how much slower exhaustive enumeration is than the algorithm given by Huffman.[1]

Each of the A_n alphabetical encodings may be converted into $n!$ of the T_n encodings by permuting its codes in all possible ways. It follows that $T_n = n!A_n$, and it suffices to prove Theorem 12. Consider for $n \geqq 2$ an exhaustive alphabetical encoding of n letters. Some number $k = 1, \ldots, n-1$ of these letters has a code with prefix 0. These k codes, each with its leading digit 0 removed, have been shown (Lemma 4) to form one of the A_k exhaustive alphabetical encodings of k letters. Similarly, the remaining $n - k$ codes, minus their leading digits 1, form one of the A_{n-k} exhaustive alphabetical encodings of $n - k$ letters. Thus, if $n \geqq 2$,

$$A_n = \sum_{k=1}^{n-1} A_k A_{n-k}, \tag{14}$$

while $A_1 = 1$. To solve (14), construct the generating function $a(x) = A_1x + A_2x^2 + A_3x^3 + \ldots$. By (14), $a(x) = x + a^2(x)$; i.e.,

$$a(x) = \tfrac{1}{2}(1 - \sqrt{1 - 4x}). \tag{15}$$

The negative sign of the square root is needed to make $a(0) = 0$. The series for $a(x)$ is obtained using the binomial theorem with power $\frac{1}{2}$. The coefficient of x^n (which is A_n) has the expression (12).

VII. ENCODINGS WITHOUT THE PREFIX PROPERTY

So far in this paper very little has been said about encodings without the prefix property. For instance, we restricted the best alphabetical encoding to be the encoding having the lowest cost among all alphabetical order-preserving encodings having the prefix property. However, in view of the fact that the special encoding given in Table I is an alphabetical encoding and has cost 4.1801, it appears to be advantageous to dispense with the prefix property requirement. However, not very much is known about the properties of encodings lacking the prefix property, and, in fact, it is not known whether the special encoding given in Table I can be further improved or not. In fact, it was not constructed on the basis of any general procedure, but was found by a heuristic method. The next few paragraphs will give a few results which we have found about encodings without the prefix property, but will also give some examples of the difficulties which it is possible to get into when using such encodings.

It should be noted that a message which begins with the letter Y in the special encoding cannot be deciphered as soon as the Y has been received, but it is necessary to wait for further received digits in order to distinguish it from a Z. In particular, in the case of the message enciphered as 11111101111110 it is necessary to wait for the 14th received binary digit before the first letter can be deciphered.

In general, we will say that the *delay of a presumed message* is d if it is necessary to wait for the receipt of the first d binary digits before the first transmitted letter can be recognized. We will say that the *delay of an encoding* is d if d is the least upper bound of the delays of all presumed messages of that encoding. We will say that an encoding has the *finite delay property* if the delay of that encoding is finite. For instance, the special encoding of Table I has the finite delay property, and in fact has delay 14.

Theorem 14: If an encoding C has infinite delay, then there exists a presumed message of C which has infinite delay.

Proof: Given an encoding C with infinite delay, there exists an infinite

sequence of presumed messages M_1, M_2, M_3, ... such that M_i has delay at least i. Then either the set of those presumed messages M_i whose first binary digit is 0 or the set whose first binary digit is 1 is an infinite set. We thus can choose an infinite subsequence of presumed messages M_1, M_2, M_3, ... such that M_i has delay at least i and such that all of the messages agree on the first binary digit. Proceeding by induction, we can choose at the kth step a subsequence of presumed messages which all agree on the first k digits. Then the infinite presumed message whose kth binary digit is the kth binary digit of all presumed messages remaining after the kth inductive step is a presumed message, and has infinite delay.

For an encoding to be useful in practice, it seems likely that it must have the finite delay property. This would permit a deciphering machine to be built having only a finite amount of memory, and it would permit two-way communication (as in telephony) to be almost instantaneous. However, in delayed communication systems (common in telegraphy) for which a tape is used for storing messages, this tape might be used to provide the unbounded amounts of memory needed to decipher an infinite delay encoding.

To investigate further the problems of designing an optimal-cost encoding of any sort (such as an alphabetical-order encoding), without requiring it to have the prefix property, it should be remarked that the problem is finite, but not necessarily easy to attack. That is, given an alphabet in which all of the letters have positive probability, and given a constant K, there are only a finite number of encodings of this alphabet which have a cost less than K. For if m is the smallest of the probabilities, there are not more than K/m digits in the longest code of any such encoding, and there are only a finite number of encodings of an n-letter alphabet in which each code has length less than K/m. However, this number would be astronomically large for any alphabet of reasonable size.

One particular way of generating encodings which will be used in a few examples below is of some general interest. The *reversal* of an encoding C is a new encoding (which will be called C^* for the remainder of this paper) which is obtained by letting the code for each letter be written in the reverse order. This interchanges the direction of increasing time, and changes many of the properties of the encoding, but it does preserve unique decipherability.

Table V demonstrates many of the properties and complications of encodings, contrasting the one having the prefix property with three other encodings lacking this property. Each of the four encodings shown

TABLE V

Letter	Probability	First Code	Second Code	Third Code	Fourth Code
A	0.330	000	00	00	00
B	0.005	001	001	0011	00111
C	0.330	01	10	01	01
D	0.005	10	101	0111	01111
E	0.330	11	11	10	10
	Cost	2.335	2.01	2.02	2.03

preserves alphabetical order, and each is uniquely decipherable. The first encoding has the prefix property, and in fact is the best alphabetical encoding in the sense used in this paper. However, it has an appreciably higher cost than either of the other three encodings, none of which has the prefix property. The reversals of each of the last three encodings have the prefix property, but the reversal of the first encoding does not.

The second encoding of Table V has the lowest possible cost of any uniquely decipherable binary encoding by Theorem 11, since it is the reversal of a Huffman encoding. However, the second encoding has infinite delay, since the presumed message 001111 . . . has infinite delay. Furthermore, the second encoding, although it preserves the alphabetical order of individual letters, does not preserve the alphabetical order of words made up out of these letters. For instance, the enciphered form of CE is a larger binary number than the enciphered form of DA, although the latter occurs later in alphabetical order. The property of preserving alphabetical order of all words will be called the *strong alphabetical property*, and it has already been shown that alphabetical encodings having the prefix property have the strong alphabetical property. However, both the alphabetical encoding and the special encoding of Table I have the strong alphabetical property, and all of the encodings of Table V except the second encoding have the strong alphabetical property. There would be very little to be gained by employing an alphabetical order encoding for sorting or dictionary purposes unless it had the strong alphabetical property.

The third encoding lacks these defects of the second encoding, but it has a special one of its own, about which more will be said in the next section. This defect has to do with synchronizing, and it can be explained in this case by the observation that every code of the third encoding has an even number of binary digits. Thus, if the deciphering circuit starts up while it is out of phase, it can never get back in phase. The two phases correspond to the odd-numbered and the even-numbered binary

digits, and the deciphering machine, if it is out of phase, would never get back in. In this case, where there are certain codes which cannot occur, the defect could be remedied by designing the circuit to additionally change phase if it ever receives a code 1011 or 1111, but this adds an extra complication to the circuit. However, the first and second encodings have the property that each of them will automatically get back in synchronism with probability 1, without the addition of any other codes or any other special features to the circuit.

The fourth encoding has none of these defects, and since its cost is so near to the least possible, it would undoubtedly be a reasonably good choice as a solution, if this particular alphabet had arisen in an actual practical problem.

So far in this paper, each example of an encoding with the finite delay property has had a delay equal to $N_{\max}$, where $N_{\max}$ is the number of digits of the longest code of the encoding. This result does not hold in general, as is illustrated by Table VI. The fifth encoding has $N_{\max} = 6$, but it has delay 8.

TABLE VI

Letter	Fifth Code	Sixth Code
W	00	00
X	001	01
Y	101	10
Z	110101	11

The encodings having the finite delay property but not the prefix property, such as the special encoding of Table I and the fifth encoding of Table VI, provide counterexamples which contradict Remark II of Schützenberger (Ref. 5, page 55) and provide the example which is asked for in the sentence following Remark I of the same paper.

As an alternative to the above method of expressing quantitatively the finite delay property, we may make the following definitions for use later in this paper. We will say that the *excess delay of a presumed message* is e if it is necessary to wait for the receipt of e binary digits beyond the end of the first transmitted letter of the presumed message before this first letter can be recognized. We will say that the *excess delay of an encoding* is e if e is the least upper bound of the delays of all presumed messages of the encoding.

If d is the delay of an encoding, e is its excess delay, and $N_{\min}$ and $N_{\max}$ are, respectively, the minimum and maximum numbers of digits of any codes of the encoding, then we obviously have $e + N_{\min} \leqq d \leqq e + N_{\max}$. Then an encoding has the finite delay property if and only if the

excess delay of that encoding is finite. Also, an encoding has the prefix property if and only if the excess delay of that encoding is 0.

VIII. SELF-SYNCHRONIZING PROPERTIES

Problems of how to make a transmitting device and a receiving device become and remain synchronized with each other are important in the engineering design of many kinds of systems. Since the encodings discussed in this paper are variable-length, it might seem that the synchronizing problem for enciphering and deciphering circuits would be especially difficult. However, the synchronizing problem is very simple for many variable-length binary encodings, because of a particularly favorable property which they possess. These remarks can best be illustrated by an example. Suppose that (using the alphabetical encoding of Table I as an example) a message beginning 1110011110100111000 . . . is received, and we wish to observe how a deciphering circuit would decipher it. Since the encoding has the prefix property, the deciphering circuit should first find a code which is a prefix of this message, and then decode this to obtain the first letter T of this message. Proceeding with

TABLE VII

:				T				:					H					:				A				:				T				:		#		:	
	1		1		1		0		0		1		1		1		1		0		1		0		0		1		1		1		0		0		0		···
		:					R					:				T				:					M					:				I				:	

the remaining part, it then finds the letter H, and then the rest of the deciphered version shown in the first line of Table VII, where the symbol ":" is used to mark the divisions between those sequences of binary digits which were deciphered as individual letters.

Next suppose that the same sequence of digits had been received, but that the deciphering circuit was not in synchronism with the enciphering circuit. In particular, suppose that, when the deciphering circuit was first turned on, it was in the state that it would be in if it were partly through the operation of deciphering some letter, and that the initial 1 of the message was interpreted as the last digit of this letter. This deciphering is indicated on the third line of Table VII. Once again, the symbol ":" has been used to mark the divisions between letters. Then these two decipherings are out of phase (i.e., out of synchronism) with one another at the beginning of the message, but at the end of the received message they are in phase with each other, as is indicated by the fact that the ":" symbols align with each other at the right end of Table VII. This means that the deciphering circuit would have automatically become synchronized, without any special synchronizing circuits or

synchronizing pulses being necessary. It was, of course, necessary for at least two of the codes of the encoding to end in the same sequence of digits, but this is very likely to happen for any variable-length encoding, unless special efforts are made to prevent it.

However, if we had been using a fixed-length encoding, such as the sixth encoding of Table VI, in which all of the codes have a fixed length k, there would be exactly k different phases in which the deciphering circuit might find itself, and the circuit could never make a transition between them. No pair of different codes can end in exactly the same sequence of digits, and so no two of these phases can become synchronized. Each of these phases will have all of the codes ending after j digit times, and after $k + j$, $2k + j$, etc., where j is the remainder obtained on dividing the position of the symbol ":" by k, and hence j can take on k different possible values.

Also, even in the case of variable-length encodings, if all of the code lengths are divisible by some integer k, then there will be at least k different phases. For if the position of one occurrence of the symbol ":" has remainder j when divided by k, the position of all other occurrences of the symbol ":" in this phase of decipherment will have the same remainder.

The above remarks apply strictly to exhaustive encodings, but may not apply where there are certain sequences of digits which can never occur. For if such a sequence of digits does occur, this may be used by the circuit as a special indication that it is out of phase, and hence it may be possible to build auxiliary circuits which can cause resynchronization, even when a fixed-length encoding is used. So a more complete treatment of synchronization would allow such auxiliary circuits, but here we will consider only self-synchronization, which is carried out inherently by the same means as is used for deciphering.

To speak more precisely about the self-syncrhonizing properties, we will make some definitions. Given any encoding C and any

> finite sequences x and y such that x is not the enciphered form (with respect to encoding C) of any message, and xy is a presumed message, (16)

if z is a finite sequence of binary digits such that both xyz and yz are complete enciphered messages, we will say that z is a *synchronizing sequence for x and y*. As an example, we have seen in Table VII that 011110100111000 is a synchronizing sequence for 1 and 110.

Given any uniquely decipherable encoding C which has some codes of length more than 1, exactly one of the three statements given below will hold:

i. For all (16), there is no z such that z is a synchronizing sequence for x and y. The encoding C will then be said to be *never-self-synchronizing.*

ii. For each (16), there is a z which is a synchronizing sequence for x and y. The encoding C will then be said to be *completely self-synchronizing.*

iii. For some (16), there is a synchronizing sequence for x and y, but for other (16), there is no synchronizing sequence for x and y. The encoding C will then be said to be *partially self-synchronizing.*

Furthermore, we will define a sequence z to be a *universal synchronizing sequence* for the encoding C if, for all (16), this same sequence z is a synchronizing sequence for x and y.

Theorem 15: Given an exhaustive encoding C, then C is completely self-synchronizing if and only if there exists a z which is a universal synchronizing sequence for C.

Proof: A universal synchronizing sequence clearly satisfies the conditions of the definition of completely self-synchronizing, so it remains only to construct a universal synchronizing sequence, given that there is a synchronizing sequence for each finite sequences x and y. By the exhaustive property, there is a code consisting entirely of 0's. We will assume that there are k 0's in this code. We will construct our z by starting with $N_{\max}$ 0's, where $N_{\max}$ is the length of the longest code of C; after this, there are only k different phases in which the circuit could be. Then we find a synchronizing sequence for two of these phases (for instance, a synchronizing sequence for 00 and 0), and put this next after our sequence. Next we put on the sequence of $N_{\max}$ 0's again. There are now at most $k - 1$ phases to synchronize, and, adding on sequences for these one at a time, we eventually construct our desired universal synchronizing sequence.

The alphabetical encoding of Table I can be shown by Theorem 15 to be completely self-synchronizing, since the sequence 010001011 is a universal synchronizing sequence for this encoding. The message AD has this sequence as its enciphered form. In addition, there are many other short universal synchronizing sequences for this encoding, such as the enciphered forms of ✱Y, AY, BD, BY, EY, HI, ID, JO, JU, MW, NY, OW, PO, PU, TY, etc. Since just these digraphs listed here occur as about three per cent of all digraphs in connected English text,[9] it can be seen that, if English text were transmitted by use of this encoding, it would be quite likely to synchronize itself very quickly.

In fact, it is easy to see that any exhaustive encoding which is completely self-synchronizing will synchronize itself with probability 1 if the messages sent have the successive letters independently chosen with

any given set of probabilities, assuming only that all of these probabilities are positive numbers. This will occur since the probability of a universal synchronizing sequence occurring at any given time is positive, and, if we wait long enough, this will have happened with probability 1.

The fact that this occurs with probability 1 does not make it quite certain to occur, and, in fact, it is possible to choose arbitrarily long sequences of English words which do not contain a universal synchronizing sequence. An example of such a sequence for the alphabetical encoding of Table I is

CHECK ✱ SYNCHRONISM ✱ OF ✱ LONG ✱ FILTHY
✱ CHUCKLE ✱ HEH ✱ HEH ✱ HEH ✱ HEH · · · ·

But such a sequence is extremely unlikely to continue indefinitely in any practical communication system or record-keeping system. Also, slight complications of the encoding could permit certain sequences which are certain to occur in English text (such as a period followed by a space symbol) to be universal synchronizing sequences.

One quality which might be worth comparing for various proposed encodings under consideration for possible use might be the average speed with which they synchronize themselves, when carrying typical traffic. This speed could be calculated from a sufficiently good knowledge of the statistics of the traffic, but it could more easily be measured experimentally, either by the use of actual enciphering and deciphering circuits, or by simulating their behavior on a digital computer.

The synchronization problem occurs not only when the equipment is first turned on, but also in transmission systems for which there is a noisy channel. For if some digits of a message encoded in a variable-length encoding are changed, the change may cause the circuits to get out of synchronism by the change of a short code into the prefix of a long one, or *vice versa*. Also, of course, temporary malfunctions of the enciphering or deciphering circuit themselves might cause them to get out of phase.

It may be of interest to enumerate the known results about combinations of synchronizing properties and lengths of the codes of exhaustive encodings.

If an exhaustive encoding has a fixed length (all codes having length the same integer k), then it must be

never-self-synchronizing. (17)

If an exhaustive encoding has all the lengths of its codes divisible by

some integer $k > 1$, but these lengths are not all equal to k, then it must be one of the following:

never-self-synchronizing, (18)

partially self-synchronizing. (19)

If an exhaustive encoding has the greatest common divisor of the lengths of its codes equal to 1, then it must be one of the following:

completely self-synchronizing, (20)

partially self-synchronizing, (21)

never-self-synchronizing. (22)

Of the above six cases, (17), (19) and (20) occur very much more commonly than the others. In fact, it is very difficult to construct examples of the other three, unless you deliberately set out to do so. The following theorems will give indications of the fact that cases (18) and (22) are hard to obtain.

Theorem 16: Given an exhaustive encoding which is never-self-synchronizing, if we let

$$Q = \sum N_i 2^{-N_i}, \tag{23}$$

then Q will always be an integer.

It can be seen that, in the case of a fixed-length code, Q will be the length. However, no one of the exhaustive encodings (except those having fixed length) listed so far in this paper has an integer value for Q. Rather than give the full details of a rigorous proof of Theorem 16, only the main ideas involved will be explained. The sum Q is the average length of the codes obtained by deciphering a presumed message, if the presumed message was obtained by choosing 0's and 1's as successive digits by independent choices having probability one-half. If we put such a random presumed message into the deciphering circuit, we have several different phases in which it may be deciphered. By the never-self-synchronizing property no two of these phases can ever come together.

Let H be the set of all prefixes of the presumed message. Then two of these prefixes will be said to be of the same phase if they are of the form θ and $\theta\Phi$, where Φ is the enciphered form of a complete message. The set H is subdivided by the equivalence relation "being of the same phase" into B distinct sets, where B is the number of phases. By symmetry, the probability that any two given members of H will be of the

same phase is equal, and, since each phase occurs with equal probability and the sum of all of them is 1, each phase occurs with probability $1/B$, where B is the number of phases. However, Q was the expected difference in length between a given member of H and its next longer member; hence, we will have $Q = B$.

Theorem 17: Given an exhaustive encoding C, C is never-self-synchronizing if and only if its reversal C^* has the prefix property.

Suppose that C is not never-self-synchronizing. By the definition of synchronizing sequence, there exist finite sequences x, y and z such that x is not the enciphered form of a message, but yz is the enciphered form of message m_1 and xyz is the enciphered form of message m_2.

For some values of n the last n letters of m_1 may agree with the last n letters of m_2. But, by the fact that x is not the enciphered form of a message, there is a largest value of n for which this is true. Let this largest value be n', and let the letters which are $n' + 1$ from the end of m_1 and m_2, respectively, be called L_1 and L_2. Then $C(L_1)$ and $C(L_2)$ are both suffixes of the same message (the previous part of xyz), and hence the reversed form of one of them is a prefix of the reversed form of the other.

The converse follows more readily, since, if θ and $\theta\Phi$ are both codes of C^*, then the reversed form of θ is a synchronizing sequence for the reversed form of Φ and the null sequence.

To return to the problem of which of cases (17) through (22) can occur, it can easily be shown by the use of Theorems 16 and 17 that, among all exhaustive encodings in which not all codes are of the same length, the only ones which are never-self-synchronizing and have fewer than 16 letters in their alphabet are the encoding which encodes a nine-letter alphabet by using the list of codes (000, 0010, 0011, 01, 100, 1010, 1011, 110, 111), and the reversal of this encoding. This encoding is due to Schützenberger.[5]

This provides an example showing that case (22) can occur. That (21) can occur is shown by an encoding (derived from the above by composition) using the list of codes (000000, 0000010, 0000011, 00001, 000100, 0001010, 0001011, 000110, 000111, 0010, 0011, 01, 100, 1010, 1011, 110, 111).

It is also possible to construct an example of case (18), but the one we have found is too complicated to be worth presenting here.

IX. ONE REALIZATION FOR ENCIPHERING AND DECIPHERING CIRCUITS

Some reluctance to use variable-length encodings has been based on the opinion[10,11] that it is hard to build circuits to encipher or decipher

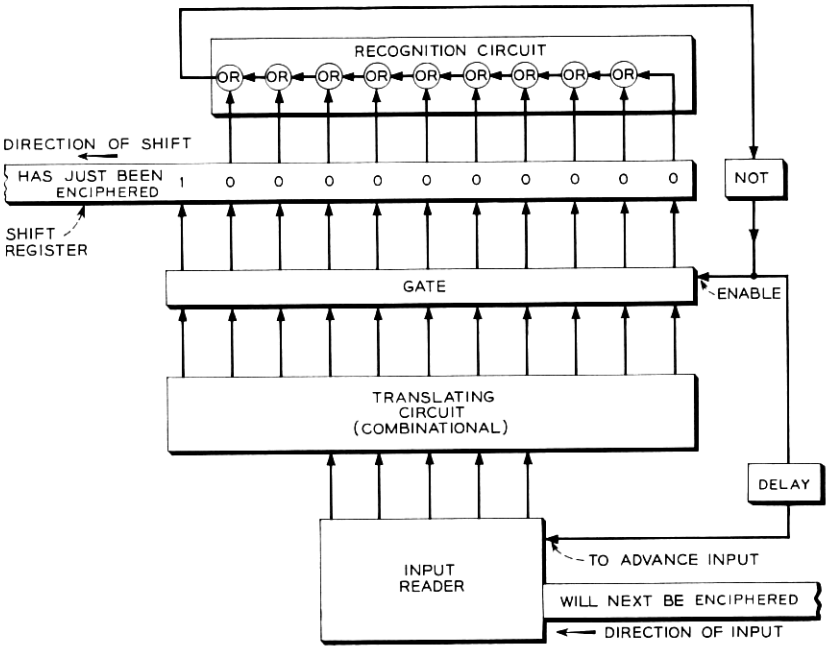

Fig. 1 — Block diagram of enciphering circuit.

them. Descriptions will be given below for one circuit for doing each of these, using principally just a shift register and a combinational translating circuit. Since using any code requires having a combinational translating circuit, and since presumably most devices using coded alphabetical information are likely to cause it to pass through a shift register, the kind of circuit described below would add very little complexity to such machines, and would automatically give them the self-synchronizing property, in the case of most variable-length binary encodings.

The enciphering circuit, shown in Fig. 1, contains a shift register containing the words "HAS JUST BEEN ENCIPHERED" followed by a binary digit 1 and a string of zeros as long as the longest code which can occur in the variable-length encoding. We will assume that it is in such a state as to have the zeros as shown, although it can easily be seen that it will get into this state if it starts in any other condition.

The circuit of Fig. 1 also contains an input reader (which can for concreteness be thought of as a punched paper tape reader, although it could be a buffer or other input device), which can read in one letter

at a time whenever it is given a pulse on the lead labelled "to advance input".

The recognition circuit, which consists of a multiple-input OR circuit followed by a negation circuit, gives an output whenever there are as many binary zeros present as there are in the illustration. This sends a signal to enable the gate, letting the code corresponding to the next letter be read into the locations previously occupied by the 1 and all of the zeros. However, the translating circuit, which translates the letters into this encoding, instead of being designed to give directly the original variable-length encoding, gives an encoding which differs from it by having an extra "1" added to the end of each code. The output of the recognition circuit also goes to advance the input, reading in the next letter to be converted, after passing through a delay sufficient to be sure that the gate is now no longer enabled. This delay prevents the letter being translated from changing while it is being gated into the output shift register.

As soon as the new code has been read into the shift register, it begins to be shifted along to the left in Fig. 1. The 1 at the end of the code serves to mark the end of the code during this shifting, but it will be eliminated from the enciphered form of the message. The shift register is connected so that, when it is shifted, a 0 appears at the right end. As soon as the 1 passes beyond the end of the recognition circuit, there will be only zeros present, and hence the recognition circuit will again recognize the end of a letter and repeat the cycle as given above.

Instead of having a counter or a special sequential circuit to keep track of where the current letter ends, this has been done here by adding a single binary digit to the code and adding one to the length of the required shift register.

Similarly, an analogous scheme can be used to decipher from a variable length code into any other representation for letters, by using one special position in the shift register, as shown in Fig. 2. This deciphering circuit can be built only for encodings having the finite delay property, although the enciphering circuit of Fig. 1 can be used for any binary encoding.

The shift register into which the digits to be deciphered are shifted is divided into two halves, which will be called the left half and the right half. The right half has e digit positions, where e is the excess delay of the encoding. The left half has $N_{\max} + 1$ digit positions, with the extra 1 being used to mark the end of those digits which already have been deciphered.

At the beginning of the cycle we will assume that the left half of the shift register has just been cleared to the state shown in Fig. 2, that is,

to contain N_{max} 0's followed by a 1. Next, the digits of the message to be enciphered shift toward the left. Since the 1 precedes them, it marks clearly how many of these digits have been shifted into the left half. As soon as all of the digits of the code of the first letter of the message have been shifted into the left half, the translating circuit will then give its outputs. It gives the translated codes for the letter, as well as giving another output, w, which equals 1 only when the complete first letter is present. The translating circuit makes use of the inputs from only the left half of the shift register, ignoring the digits in the right half, unless the code C present in the left half is a code which is also a prefix of another code. It makes use only of those digits from the right half which are necessary to distinguish between this code and the partially shifted-in code of which it is a prefix. It gives the output $w = 1$ whenever the entire code for the first letter of the enciphered message has been shifted over into the left half and, whenever only a prefix of the code of the first letter is there, the output w will equal zero.

This output w will then cause the left half to clear back to its original state, and, after a delay sufficient to allow the output to be received, it gives the "to advance output" signal to the output punch or buffer.

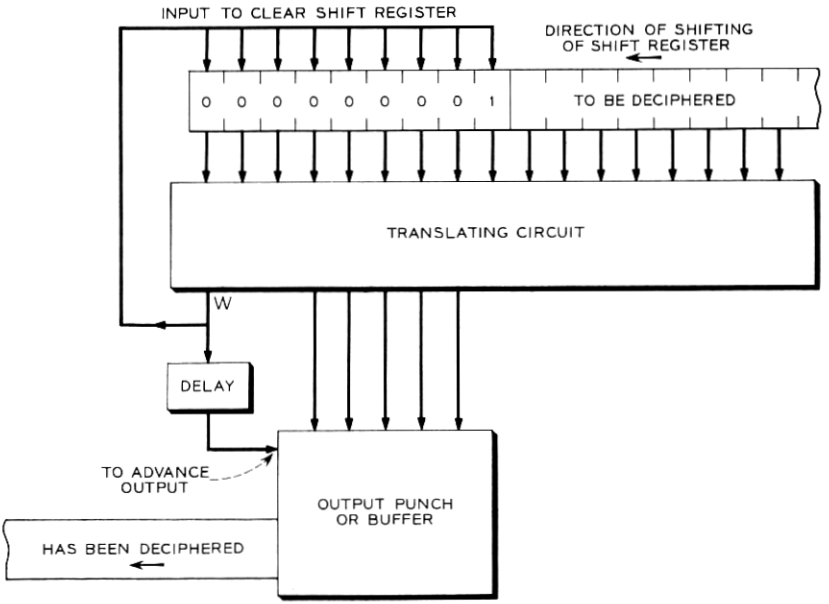

Fig. 2 — Block diagram of deciphering circuit.

The deciphering circuit then repeats the above cycle for the next letter of the message.

The translating circuit of this deciphering circuit must give the appropriate outputs whenever the complete code for the first letter is present in the left half of the shift register, and must give $w = 1$ in these cases. It must also be designed to give the output $w = 0$ whenever an incomplete prefix of the first letter is present, but, since in general there may be many states of the shift register which do not correspond to either a letter or a prefix, there may be many "don't cares" occurring in the design of this translating circuit, which will permit it to be simpler than a completely specified function having this many inputs.

The time delay between the receipt of the beginning of an N-digit code for a letter and the actual sending of this letter to the output punch or buffer will be $N + e$, which may sometimes be slightly longer than the delay d of the message. However, the circuit for doing the deciphering in the minimum time would be more complicated, in that it would not always clear the shift register to the same state, so it is not presented here.

However, in the enciphering circuit given in Fig. 1 there is only a delay of one digit time, while the message is shifted through the one extra stage at the left end of the shift register. Hence, neither of these two circuits operates in quite the minimum possible time, since speed has been sacrificed for simplicity of construction.

X. FURTHER PROBLEMS

There are many further problems suggested by the ideas discussed in this paper, and which we have not been able to solve. Are there any binary encodings which satisfy (9) other than the exhaustive encodings and their reversals? Are there any encodings C which satisfy (9) and such that both C and its reversal C^* have the finite delay property without both C and C^* having the prefix property? Given an encoding which is uniquely decipherable but which does not possess the finite delay property, does the set of presumed messages having infinite delay always form a finite set? Does it always form a set of measure zero? Is there a simple polynomial in $N_{\max}$ and n which will be an upper bound to the delay of any encoding having the finite delay property? Are the encodings for which the algorithm of Sardinas and Patterson[2] fails to terminate precisely the same as the encodings having infinite delay? Given any encoding having infinite delay, is there a Turing machine (perhaps having several tapes and several reading and writing heads on

each) which can decipher any K-digit message in a length of time which is less than a constant times K?

XI. ACKNOWLEDGMENTS

We would like to express our thanks to T. H. Crowley for suggesting Theorem 7, and to S. P. Lloyd for suggesting to us some of the complications of encodings which do not have the prefix property.

REFERENCES

1. Huffman, D. A., A Method for the Construction of Minimum-Redundancy Codes, Proc. I.R.E., **40,** September 1952, p. 1098.
2. Sardinas, A. A. and Patterson, G. W., A Necessary and Sufficient Condition for Unique Decomposition of Encoded Messages, I.R.E. Conv. Rec., 1953, Part 8, p. 104.
3. McMillan, B., Two Inequalities Implied by Unique Decipherability, I.R.E. Trans., **IT-2,** December 1956, p. 115.
4. Mandelbrot, B., On Recurrent Noise Limiting Coding, Proceedings of the Symposium on Information Networks, Polytechnic Institute of Brooklyn, April 1954, p. 205.
5. Schützenberger, M. P., On an Application of Semi-Groups Methods to Some Problems in Coding, I.R.E. Trans, **IT-2**, September 1956, p. 47.
6. Michel, W. S., Fleckenstein, W. O. and Kretzmer, E. R., A Coded Facsimile System, I.R.E.-Wescon Conv. Rec., 1957, Part 2, p. 84.
7. Dewey, G., *Relativ Frequency of English Speech Sounds*, Cambridge Univ. Press, Cambridge, Eng., 1923, p. 185.
8. Shannon, C. E., A Mathematical Theory of Communication, B.S.T.J., **27,** July 1948, p. 379; October 1948, p. 623.
9. Pratt, F., *Secret and Urgent*, Doubleday & Co., Garden City, N. Y., 1939.
10. Brooks, F. P., Ph.D. thesis, Harvard Univ., May 1956.
11. Brooks, F. P., Multi-Case Binary Codes for Non-Uniform Character Distributions, I.R.E. Conv. Rec., 1957, Part 2, p. 63.