TABLE VI
COMPUTER RESULTS

n	r	Number of Inequivalent Codes	Number of Extremal Codes
28	7	31	14
28	13	5	5
32	5	310	239
32	7	18	16
36	11	6	0
36	17	13	1
40	13	165	4
40	19	31	11

$\beta^{13} = -\beta^0$, $\gamma^0 = 0212011222020$, $\gamma^1 = 0012102211101$, $\gamma^2 = 2002120112220$, $\gamma^4 = 2020021201122$, $\gamma^5 = 1101001210221$, $\gamma^{22} = 0112220200212$, $\delta^0 = 0020222110212$, $\delta^{11} = 2100101112201$, $\delta^{12} = 2002022211021$, $\delta^{13} = -\delta^0$, $\delta^{16} = 0222110212002$, $\delta^{17} = 1112201210010$, and $\delta^{22} = 0212002022211$.

b) $r = 19$, $c = f = 2$; $\mathscr{C} = C(\sigma) \oplus E_1(\sigma)$ where gen $C(\sigma) = \begin{bmatrix} \mathbf{1} & \mathbf{0} & 1 & 1 \\ \mathbf{0} & \mathbf{1} & 1 & 2 \end{bmatrix}$ and gen $E_1(\sigma)^* = [\alpha^0 \mu^i]$ with $\alpha^0 = 0222222222222222222$ and μ^{9841}. There are 11 inequivalent codes for the values $i = 1, 5, 19, 29, 49, 59, 65, 83, 97, 173,$ and 259. Here $\mu^1 = 0201020101101001020$, $\mu^5 = 0012112110101100120$, $\mu^{19} = 0210211010122020210$, $\mu^{29} = 2021201211222012021$, $\mu^{49} = 2120021222101100010$, $\mu^{59} = 2212100022021121212$, $\mu^{65} = 1211001220010221002$, $\mu^{83} = 0001202120121201102$, $\mu^{97} = 1210221102001112112$, $\mu^{173} = 2112120222211002021$, and $\mu^{259} = 0220020022220202000$.

IV. CONCLUSION

We conclude with a few remarks. First, the case $r = 5$ when $n = 40$ (case g) of Theorem 3) was not done. In that case, $C(\sigma)\Phi$ is $E_4 \oplus E_4$ where E_4 is the $[4, 2, 3]$ tetracode (see [12]); but the number of possible codes for $E_1(\sigma)^*$ is extremely large. However, $5 \nmid |G^*(C(\sigma)\Phi) \cap \mathscr{S}_{8,0}|$ in that case as well, and hence, the techniques of Section II on equivalence can be applied. Second, the equivalence or inequivalence of two extremal codes of length n constructed from two different values of r is still an open problem. Because the general question of when two codes are equivalent is so difficult, the power of results such as Theorem 2 becomes clear when by computer, it was relatively easy and quick to decide that the 239 $[32, 16, 9]$ codes with $r = 5$ are inequivalent. Third, the author was very surprised that the number of extremal codes found was so large. Also the high percentage of codes examined that turned out to be extremal was a surprise. This is illustrated by Table VI. In this table, "Number of inequivalent codes" refers to the number of equivalence classes of codes that were examined by computer; here the equivalence classes were those determined as if Theorem 2 held. We checked the general forms that were given in Theorems 4–7. A similar table appears in [5] for quaternary codes, and we see by comparing these two tables that the percentage of extremal codes in the ternary case is much higher. Codes of length 40, with $r = 5$, and 44, with $r \geq 5$, might be interesting to examine if it becomes computationally feasible. Fourth, one might ask if the Pless symmetry code of length 36 is the unique extremal code of that length. Finally, the programming required for this paper was done on an AT & T 6300 in Pascal.

REFERENCES

[1] J. H. Conway, V. Pless, and N. J. A. Sloane, "Self-dual codes over GF(3) and GF(4) of length not exceeding 16," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 312–322, 1979.

[2] J. H. Conway and V. Pless, "On primes dividing the group order of a doubly-even (72, 36, 16) code and the group order of a quaternary (24, 12, 10) code," *Discrete Math.*, vol. 38, pp. 143–156, 1982.

[3] ——, "Monomials of orders 7 and 11 cannot be in the group of a (24, 12, 10) self-dual quaternary code," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 137–140, Jan. 1983.

[4] W. C. Huffman, "On the [24, 12, 10] quaternary code and binary codes with an automorphism having two cycles," *IEEE Trans. Inform. Theory*, vol. 34, pp. 486–493, May 1988.

[5] ——, "On extremal self-dual quaternary codes of lengths 18 to 28, I," *IEEE Trans. Inform. Theory*, vol. 36, pp. 651–660, May 1990.

[6] ——, "On extremal self-dual quaternary codes of lengths 18 to 28, II," *IEEE Trans. Inform. Theory*, vol. 37, pp. 1206–1216, July 1991.

[7] ——, "On the equivalence of codes and codes with an automorphism having two cycles," *Discrete Math.*, vol. 83, 265–283, 1990.

[8] N. Ito, J. S. Leon, and J. Q. Longyear, "Classification of 3-(24, 12, 5) designs and 24-dimensional Hadamard matrices," *J. Combinat. Theory A*, vol. 31, pp. 66–93, 1981.

[9] C. W. H. Lam and V. Pless, "There is no (24, 12, 10) self-dual quaternary code," *IEEE Trans. Inform. Theory*, vol. 36, pp. 1153–1156, Sept. 1990.

[10] J. S. Leon, V. Pless, and N. J. A. Sloane, "On ternary self-dual codes of length 24," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 176–180, 1981.

[11] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*. Amsterdam: North-Holland, 1977.

[12] C. L. Mallows, V. Pless, and N. J. A. Sloane, "Self-dual codes over GF(3)," *SIAM J. Appl. Math.*, vol. 31, pp. 649–666, 1976.

[13] V. Pless, "Symmetry codes over GF(3) and new five-designs," *J. Combinat. Theory*, vol. 12, pp. 119–142, 1972.

[14] V. Pless, N. J. A. Sloane, and H. N. Ward, "Ternary codes of minimum weight 6 and the classification of the self-dual codes of length 20," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 305–316, 1980.

[15] V. Pless, *Introduction to the Theory of Error-correcting Codes*, second ed. New York: John Wiley, 1989.

[16] V. Y. Yorgov, "A method for constructing inequivalent self-dual codes with applications to length 56," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 77–82, Jan. 1987.

A Fast On-Line Adaptive Code

Boris Ya. Ryabko

Abstract—**There are two classes of data compression algorithms. One class has redundancy log log $n + 0(1)$, where n is the alphabet size, and an encoding time $0(\log^2 n)$, $n \to \infty$. The other has redundancy $0(1)$ and an encoding time $0(n)$. A code is presented combining advantages of both classes of compression methods: its redundancy is $0(1)$ and the encoding and decoding time is $0(\log^2 n)$ per letter, which is close to the lower bound $0(\log n)$.**

Index Terms—**On-line adaptive coding, Huffman code, encoding, book-stack method.**

INTRODUCTION

The first adaptive on-line code was apparently proposed in 1980 [14], and then rediscovered in the papers of Bentley, Sleater, Tarjan

Manuscript received January 17, 1990.

The author is with Applied Mathematics and Cybernetics, Novosibirsk Institute of Communication, Kirov Street 86, Novosikizsk-125, Russian Federation.

IEEE Log Number 9108027.

and Wei [2] in 1986, and Elias [3] in 1987; see also [15]. (In all these papers, this code [3] is named differently: e.g. "book stack" in [14], "move-to-front scheme" in [2], and "recency rank coding" in [3]). In [4], Gallager proposed an efficient adaptive on-line scheme of Huffman coding, which was later modified by Knuth [7]. On-line coding methods deal with the problem of encoding arbitrary words $x_1 x_2 \cdots x_N \geq 1$, in a finite alphabet $A = \{a_1, \cdots, a_n\}$, $n \geq 2$. The code of a symbol x_i, $1 \leq i \leq N$, in on-line coding may depend on $x_1 \cdots x_{i-1}$ but should be independent of $x_{i+1} \cdots x_N$. The code efficiency is estimated, first by the degree of compression, second by the encoding and decoding time per letter, and third by the storage capacity of the encoder and decoder (when they are implemented by a computer). We will estimate the degree of compression of a code by the redundancy, defined as the difference between the mean codewords length in given code and the mean codeword length in a Huffman code based on the presented frequencies of letters. The storage capacity of the encoder and decoder will be evaluated in bits. The encoding time will be evaluated by the maximum number of operations with one bit words used for encoding and decoding of a symbol, the maximum being taken over all symbols in all possible words of the alphabet A. Formally, the encoder and decoder could be considered as implemented on a RAM computer [1] that is the model of a "conventional" computer. The main objective will be to determine efficiency estimates asymptotic in n, the number of symbols in the alphabet. Since n is quite large $(2^8 - 2^{32})$ for the file compression on computers.

There are several characteristics of known on-line codes. Gallager's [4] and Knuth's [7] adaptive Huffman code has been improved by a number of authors. Vitter [20] found the optimal adaptive Huffman code, and Jones [11] proposed a data structure to reduce the encoding and decoding time. The redundancy of an adaptive Huffman code does not exceed a constant, independent of the alphabet size, n and the maximum encoding and decoding time is $0(n)$. Therefore, for $P(a) = 2^{-n}$, the codeword length for the symbol a will be equal to n bits, and the encoding and decoding time, thus equal to $0(n)$ (because it is necessary to scan a codeword both while encoding and decoding). Looking ahead, we note that reduction of the maximum encoding and decoding time is only possible when using a code having a maximum codeword length that does not much exceed log n.

For the "book stack" method, the redundancy, when the number of symbols, n, in the alphabet is large, equals log log n + 0(1). In [2], a simple implementation of "book stack" is proposed when the maximum time of encoding and decoding is 0(log n). Note that this value is fairly close to the obvious lower bound, 0(log n). Elias [3] also proposed interval encoding. It has the same asymptotic characteristics as "book stack" encoding (see Table I).

For file compression, an arithmetic code proposed and studied in [12], [13], [22] is very popular. However, in both arithmetic and adaptive Huffman codes, the codeword length of rarely occurring symbols can achieve n + 0(1) bits. Hence, the maximum time for encoding a symbol is proportional to n. Starkov [18] offered 'a generalized code' having the characteristics listed in Table I.

The storage capacity of the encoder and decoder for all the codes mentioned is asymptotically equal to $0(n \log n)$ bits (for $n \to \infty$). Some have relatively large redundancy, log log n + 0(1), and asymptotically small time required for encoding and decoding, $0(\log^2 n)$.

Others, alternatively, have a redundancy that is asymptotically minimal, 0(1), but longer encoding/decoding time, $0(n)$.

The code presented in this correspondence is called a frequency code in which both the redundancy and time of encoding and decoding a symbol are small. Its characteristics are listed in the last line of Table I. The memory storage requirement of this code is asymptotically the same as for the other codes discussed above. The construction of the frequency-code is based on alphabetic Gilbert-Moore code [6] and on the results of Krichevsky [8], [9].

Besides the forementioned codes for file compression, there are other methods that are not symbol-to-word. First, there are methods based on the Ziv–Lempel scheme [23] that are considered in detail in [19]. Second, there are the universal coding methods reviewed in [10], as well as observation-based coding methods [5], [8], [9] that are conceptually similar to Willems code [21] combining both observation-based coding concepts and interval encoding. These methods are not considered in the present correspondence, since they are applied to the coding of blocks and require the storage capacity of the encoder and decoder to be significantly greater than for symbol-to-word coding.

The main results of the correspondence have been announced in [16].

II. Frequency Code

First, we present a brief informal definition. A frequency code, sometimes denoted as ω, is designed for coding words $x_1 x_2 \cdots x_N$ of alphabet $A = \{a_1, \cdots, a_n\}$. Prior to coding a symbol x_i, $1 \leq i \leq N$, the occurrence rate of all symbols of the alphabet A is counted in the "window" of length w, i.e., in the word $x_{i-w} x_{i-w+1} \cdots x_{i-1}$, where w is a parameter. According to the frequencies found, the code is then constructed as an approximation to the alphabetic Gilbert–Moore code [6], and x_i is encoded by this code. When decoding, the same operations are repeated. The main problem is associated with performing the encoding and decoding operations in the time $0(\log^2 n)$.

Let

$$l = \lceil \log n \rceil, \qquad L = 2^l. \tag{1}$$

During the coding according to method ω, we will consider only the window of length w for

$$w = (2^r - 1)L, \tag{2}$$

where r is an integer. For a description of ω, it is convenient to extend a word $x_1 x_2 \cdots x_N$ by w symbols to the left as $\cdots a_2 a_1 a_n \cdots a_3 a_2 a_1 a_n a_{n-1} \cdots a_2 a_1 x_1 x_2 \cdots x_N$. The word obtained will be expressed as $x_{-w+1} x_{-w+2} \cdots x_{-2} x_{-1} x_0 x_1 x_2 \cdots x_N$. (This allows introduction of the window for encoding the first symbols $x_1 x_2, \cdots$, thus, simplifying our description.) For each symbol $a \in A$ and $1 \leq i \leq N$, let $P(a, i)$ be the number of occurrences of a in the word $x_{i-w} x_{i-w+1} \cdots x_{i-1}$. Define $\hat{P}$, as

$$\begin{cases} \hat{P}(a_j, i) = P(a_j, i) + 1; & i = 1, \cdots, n; \qquad 1 \leq i \leq N, \\ \hat{P}(a_j, i) = 1, & \text{for } j = n+1, \cdots, L \end{cases} \tag{3}$$

$$\begin{cases} Q(a_j i) = 2 \sum_{k=1}^{j-1} \hat{P}(a_k, i) + \hat{P}(a_j, i) \\ m(a_j i) = l + r + 1 - \lfloor \log \hat{P}(a_j, i) \rfloor; \qquad j = 1, \cdots, n; \\ \qquad\qquad\qquad\qquad\qquad\qquad i = 1, \cdots, N \end{cases} \tag{4}$$

l is given in (1), r is a parameter related to the window length, see (2). Symbol x_i encoded by a binary word that consists of the first $m(x_i, i)$ symbols from the binary expansion of $Q(x_i, i)$. Thus, the

TABLE I
CHARACTERISTICS OF KNOWN ADAPTIVE ON-LINE CODES*

N	Code	Redundancy	Time of Encoding and Decoding a Symbol
1	Book Stack [2], [3], [14]	$\log\log n + O(1)$	$O(\log^2 n)$
2	Adaptive Huffman Code [4], [7], [20], [11]	$O(1)$	$O(n)$
3	Interval Code [3]	$\log\log n + O(1)$	$O(\log^2 n)$
4	Arithmetic Code [12], [13], [22]	$O(1)$	$O(n)$
5	Generalized Shannon Code [18]	$O(1)$	$O(n \log n)$
6	Frequency Code	$O(1)$	$O(\log^2 n)$

* n is the number of symbols in the alphabet, $n \to \infty$.

length of a frequency code codeword satisfies the equality

$$|\omega(x_i/x_1 \cdots x_{i-1})| = m(x_i, i).$$

Note, that this code is the same as the alphabetic Gilbert-Moore code [6] made for the alphabet $A = \{a_1, \cdots, a_n\}$ with the set of probabilities $(P(a_j, i) + 1)/(2^r L)$, $j = 1, \cdots, n$. (From the definition of $P(a_j, i)$ and (2), it follows that the sum of these probabilities is equal to 1.) The alphabetic code is decipherable. The main objective is to achieve the fast execution of encoding and decoding.

To begin with, we will describe the encoding. When encoding a symbol x_i from a word $x_1 x_2 \cdots x_N \in A^*$, it is necessary 1) to calculate the $Q(x, i)$'s that are given in (4) in order to form the codeword based on them, and 2) alter the frequencies $P(a, i+1)$, $a \in A$, because of the window shift, according to the formulae:

$$\begin{cases} \hat{P}(x_i, i+1) = \hat{P}(x_i, i) + 1, \\ \hat{P}(x_{i-L}, i+1) = \hat{P}(x_{i-L}, i) - 1, \\ \hat{P}(a, i+1) = \hat{P}(a, i), & \text{for } a \neq x_i, \quad a \neq x_{i-L}. \end{cases} \tag{5}$$

To perform these operations quickly, a computer memory has to store $(2L - 1)$ numbers defined by the equalities:

$$D_i^1(j) = \hat{P}(a_j, i); \qquad j = 1, \cdots, L; \qquad i = 1, \cdots, N \tag{6}$$

(for $j = n+1, \cdots, L$, defined $\hat{P}(a_j, i) = 1$), and

$$D_1^{m+1}(j) = D_i^m(2j-1) + D_i^m(2j); \qquad j = 1, \cdots, 2^{l-m+1}; \quad m = 1, \cdots, l-1. \tag{7}$$

We now show how to compute $\Sigma_{m=1}^k \hat{P}(a_m, i)$, for any $k = 1, \cdots, n$, using a set $D_i^m(j)$; $m = 1, \cdots, 1$; $j = 1, \cdots, 2^{l-m+1}$; $i = 1, \cdots, N$, in $0(\log^2 n)$ operations with bits. Let the binary expansion of k be

$$k = \tau_l 2^{l-1} + \tau_{l-1} 2^{l-2} + \cdots + \tau_1 2^0, \tag{8}$$

where $\tau_s = 0$ or 1 for $s = 1, 2, \cdots, 1$. Define

$$t_m = \sum_{s=0}^{l-m} \tau_{l-s} 2^{l-m-s}, \qquad m = 1, \cdots, l. \tag{9}$$

Then, the following equality follows from (6)-(9),

$$\sum_{m=1}^{k} \hat{P}(a_m, i) = \sum_{s=1}^{l} \tau_s D_i^s(t_s), \tag{10}$$

The right sum in (10) consists of no more than 1 components, for any $1 \leq k \leq n-1$. That allows us to simplify essentially the computation of the sum in (4). Thus, using the set $\{D_i(j)\}$, we are able to form a codeword, according to (4), with no more than one summation of numbers that do not exceed $w + L$ each. (This follows from the fact that $\Sigma_{j=1}^n P(a_j, i) = w$.) From (3), it follows that $\Sigma_{j=1}^L \hat{P}(a_j, i) = w + L$. Since $w + L = 2^r L$ (see (2)), then to form a codeword according to (4), it is necessary to sum no more than one number that does not exceed $2^r l$. Hence, no more than $l(r + l)$ operations with one-bit words are required. To form t_m according to (9), we need I operations for $i = 1, 2, \cdots, l$, i.e., no more than l^2 operations with one-bit words. In addition to forming a codeword by (4), it is necessary to alter the frequencies $\hat{P}(a, i)$ according to (5) and, respectively, alter the sets $\{D_i^m(j)\}$, in (6), (7). So, we need to alter two numbers from the set $D_i^1(j)$; $j = 1, \cdots, L$ (add 1 to one number and subtract 1 from other), then two numbers from $D_i^j(j)$; $j = 1, \cdots, L/2$, up to $D_i^{l-1}(j)$; $j = 1, 2, \cdots$. So $2(l-2)$ summation operations and, hence no more than $2l(l + r)$ operations with one-bit words are required. Thus, the total number of operations for the encoding of a single symbol is proportional to $l(l + r)$.

Before the description of the decoding, consider an example. Assuming $A = \{a_1, a_2, \cdots, a_8\}$, we want to encode the word $x_1 x_2 \cdots x_{10} = a_6 a_6 a_6 a_7 a_6 a_6 a_6 a_6 a_6 a_6$. Here, $L = 8$, $l = 3$. Assume the window length $w = 8$, i.e., $w = L$, $r = 1$, see (2). Consider the coding of symbol x_{10}. There are one a_7 and seven a_6 in the window, so $P(a_1, 10) = P(a_2, 10) = \cdots = P(a_5, 10) = 0$, $P(a_7, 10) = 1$, $P(a_6, 10) = 7$, $P(a_8, 10) = 0$. According to (3) and (6), (7), we have $\hat{P}(a_1, 10) = \hat{P}(a_2, 10) = \cdots = \hat{P}(a_5, 10) = \hat{P}(a_8, 10) = 1$; $\hat{P}(a_6, 10) = 8$, $\hat{P}(a_7, 10) = 2$; $D_{10}^1(1) = D_{10}^1(2) = \cdots = D_{10}^1(5) = D_{10}^1(8) = 1$, $D_{10}^1(6) = 8$, $D_{10}^1(7) = 2$; $D_{10}^2(1) = D_{10}^2(2) = 2$; $D_{10}^2(3) = 9$; $D_{10}^2(4) = 3$; $D_{10}^3(1) = 4$; $D_{10}^3(2) = 12$. To form the codeword of symbol x_{10}, we need to find $Q(a_6, 10)$, see (4), which, in turn, requires computing $\Sigma_{m=1}^5 \hat{P}(a_m, 10)$. From (10),

$$\sum_{m=1}^{5} \hat{P}(a_m, 10) = \sum_{s=1}^{3} t_s D_{10}^s(t_s). \tag{11}$$

For $k = 5$, (8) and (9) yield $\tau_3 = 1$, $\tau_2 = 0$, $\tau_1 = 1$,

$$t_1 = \tau_3 2^2 + \tau_2 2^1 + \tau_1 2^0 = 5, \; t_2 = \tau_3 2^1 + \tau_2 2^0 = 2,$$

$$t_3 = \tau_3 2^0 = 1.$$

From this and (11), we obtain

$$\sum_{m=1}^{5} \hat{P}(a_m, 10) = 1 \cdot D_{10}^1(5) + 0 \cdot D_{10}^2(2) + 1 \cdot D_{10}^3(1) = 1 \cdot 1 + 1 \cdot 4 = 5.$$

Then, using one multiplication by 2 and one addition operation, we have

$$Q(a_6, 10) = 25 + 8 = 18 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

$$m(a_6, 10) = 1 + 1 + 3 - \lfloor \log \hat{P}(a_6, 10) \rfloor = 2.$$

Consequently, $\omega(a_6/x_1 \cdots x_9) = 10$. (Here, 10 is binary number.)

Now we shall describe the decoding procedure. Assume that $x_1 x_2 \cdots x_N$ was encoded using the frequency code as $\omega(x_1)\omega(x_2/x_1) \cdots \omega(x_N/x_1 \cdots x_{N-1})$, the symbols $x_1, \cdots, x_{i-1}$ being already decoded based on the prefix $\omega(x_1) \cdots$

$\omega(x_{i-1}/x_1 \cdots x_{i-2})$, and it is necessary to decode x_i, $i = 1, 2, \cdots, N$. Both in encoding and decoding, the values of $\{D_i^m(j); m = 1, \cdots, l; j = 1, \cdots, 2^{l-m+1}\}$ are formed. In order to find x_i, the decoder reads out $l + r + 1$ symbols of the word $\omega(x_i/x_1 \cdots x_{i-1}) \cdots \omega(x_N/x_1 \cdots x_{N-1})$. These binary characters can be denoted as $u_1u_2 \cdots u_{r+l+1}$. Using them, the following number is formed

$$u = 2^{l+r}u_1 + 2^{l+r-1}u_2 + \cdots + 2^0u_{l+r+1}. \tag{12}$$

After that, a number m is found, such that

$$2\sum_{j=1}^{m-1} \hat{P}(a_j, i) \le u < 2\sum_{j=1}^{m} \hat{P}(a_j, i). \tag{13}$$

Then the ith symbol in the encoded word $x_1 \cdots x_N$ has to be a_m, i.e., $x_i = a_m$. (This follows from (4): the difference between u and $Q(a_m, i)$ is not more than $\hat{P}(a_m, i)$).

It remains to show how to "quickly" find m that satisfies (12) using the set of $\{D_i^m(j)\}$ values. First, the inequality $u \le 2D_i^l(1)$ is verified. If this inequality holds, then $m \le L/2$. Otherwise, $m \le L/2$. Then, for $m \le L/2$, the inequality $u < 2(D_i^l(1) + D_i^{l-1}(3)$ is verified, and, for $m > L/2$, the sum $D_i^l(1) + D_i^{l-1}(3)$ is calculated and the inequality $u < 2(D_i^l(1) + D_i^{l-1}(3))$ is verified. This verification makes clear which of the four intervals $-(0, L/4], (L/4, L/2], (L/2, 3/4L], (3/4L, L] - u$ belongs to.

Proceeding similarly, we can see that in one steps a number, m will be found that satisfies (13). During each step, one comparison is performed and maybe one or two summations of numbers. Each of these numbers does not exceed 2^{l+r+1} and so the total number of operations with one-bit words is proportional to $l(l + r)$. After that, the decoder defines the codeword $\omega(x_i/x_1 \cdots x_{i-1})$, the same way as in encoding. Then, it becomes clear which characters in $u_1u_2 \cdots u_{l+r+1}$ constitute $\omega(x_i/x_1 \cdots x_{i-1})$ and which belong to $\omega(x_{i+1}/x_1 \cdots x_i) \cdots \omega(x_N/x_1 \cdots x_{N-1})$. The procedure for computing u according to (12) requires $(l + r + 1)$ one-bit operations. Thus, the total decoding time is proportional to $l(l + r)$.

For illustration, return to the previous example and consider the decoding of symbol x_{10} in the word $a_6a_6a_6a_7a_6a_6a_6a_6a_6 \cdots$. The decoder reads out $l + r + 1 = 5$ bits of the word $\omega(x_{10}/x_1 \cdots x_9)\omega(x_{11}/x_1 \cdots x_{10}) \cdots$ in order to decode x_{10}. It was shown previously, that $\omega(x_{10}/x_1 \cdots x_9) = 10$, therefore, those bits that are read out we denote as $10V_1V_2V_3$ where $V_1V_2V_3$ are arbitrary binary characters. Then, u is computed (see (12)) as

$$u = 2^{3+1} \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot V_1 + 2^1 \cdot V_2 + 2^0 \cdot V_3.$$

Obviously, for any $V_1V_2V_3$, $16 \le u \le 23$. Further, u and $2D_{10}^3(1) = 2 \cdot 4 = 8$ are compared. Since $u > 8$, then m satisfies $4 < m \le 8$. Further, $2(D_{10}^3(1) + D_{10}^2(3)) = 2(4 + 9) = 26$ is computed. Since $u < 26$, then $4 < m \le 6$. After that, $2(D_{10}^3(1) + D_{10}^1(5)) = 2(4 + 1) = 10$ is computed. Since $u > 10$, then $5 < m \le 6$, i.e., $m = 6$, and, consequently, $x_{10} = a_6$.

An estimate of the storage capacity required for computer implementation of the encoder and decoder is now provided. In both cases, when handling a symbol x_i, we need to store w symbols $x_{i-w}x_{i-w+1} \cdots x_{i-1}$, each requiring 1 bits for recording. It is necessary to store numbers $\{D_i^m(j); m = 1, \cdots, l; j = 1, \cdots, 2^{l-m+1}\}$ ($2L - 2$ numbers in total), each being stated as a word of $(1 + r + 1)$ bits. Thus, $w \cdot l + 2L(l + r)$ bits are needed for the storage of these numbers. From this and (2), it follows that total storage capacity is proportional to $2^rL \cdot l$ bits. So the following statement can be formulated concerning the complexity of a frequency code.

Statement: Let $A = \{a_1, \cdots, a_n\}$, $n \ge 2$, be an alphabet, $x_1x_2 \cdots x_N \in A^N$, $N \ge 1$, and suppose that the word $x_1 \cdots x_N$ is encoded by the frequency code having the window of length $w = (2^r - 1)2^{\lceil \log n \rceil}$ where r is an integer. Then, the maximum encoding and decoding time of a symbol does not exceed $c \cdot \log n$ ($\log n + r$), where c is a constant. The storage capacity requirement of the encoder and decoder does not exceed $2^rn \cdot \log n$.

III. The Redundancy of the Frequency Code

Let $x = x_1x_2 \cdots x_N$ be a word in the alphabet A and let φ be an on-line code. Let

$$c(\varphi, x) = N^{-1}\sum_{i=1}^{N} |\varphi(x_i/x_1x_2 \cdots x_{i-1})|$$

be the cost of using code φ to encode the word x. (Here, $|y|$ represents the length of a word y). Let μ be a probability measure on the alphabet A. The average cost of φ with respect to μ, is defined by the equality

$$\bar{c}(\varphi, \mu) = \overline{\lim_{N\to\infty}} N^{-1}\sum_{u\in A^N} \mu(u)c(\varphi, u),$$

were A^N is the set of words of length N in the alphabet A, and $\mu(x_1x_2 \cdots x_N) = \mu(x_1)\mu(x_2) \cdots \mu(x_N)$. Denote as $\bar{c}(\mu)$ the cost (average codeword length) of Huffman code constructed for the alphabet A with probability distribution μ. Define

$$\bar{r}(\varphi, \mu) = \bar{c}(\varphi, \mu) - \bar{c}(\mu), \qquad \bar{R}(\varphi) = \sup \bar{r}(\varphi, \mu),$$

where the supremum is taken over all Bernoulli sources.

The next theorem characterizes the properties of the code ω.

Theorem: Let $A = \{a_1, \cdots, a_n\}$ be an alphabet and ω the frequency code having the window $(2^r - 1)2^{\lceil \log n \rceil}$. For any Bernoulli source generating symbols from A, the redundancy does not exceed $2 + \log e/(2^r - 1)$:

$$\bar{R}(\omega) < 2 + \log e/(2^r - 1).$$

The proof is provided in [17]. The analogous result is also correct for redundancy defined for individual sequences ([17]).

References

[1] A. V. Aho, J. E. Hopkroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1976.

[2] J. L. Bently, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "Locally adaptive data compression scheme," *Comm. ACM*, vol. 29, pp. 320–330, 1986.

[3] Elias P., "Interval and recency rank source coding: Two on-line adaptive variable-length schemes," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 3–10, Jan. 1987.

[4] R. G. Gallager, "Variations on theme by Huffman," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 668–674, Sept. 1978.

[5] E. N. Gilbert, "Coding based on inaccurate source probabilities," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 304–314, Sept. 1971.

[6] E. N. Gilbert and E. F. Moore, "Variable-length binary encodings," *Bell Syst. Tech. J.*, vol. 38, pp. 933–967, 1959.

[7] D. E. Knuth, "Dynamic Huffman coding," *J. Algorithms*, vol. 6, pp. 163–180, 1985.

[8] R. E. Krichevsky, "The connection between the redundancy and reliability of information about the source," *Probl. Inform. Transm.*, vol. 4, no. 3, pp. 48–57, 1968 (in Russian).

[9] ——, "Optimal source-coding based on observation," *Probl. Inform. Transm.*, vol. 11, no. 1, pp. 37–42, 1975 (in Russian).

[10] R. E. Krichevsky and V. K. Trofimov, "The performance of universal encoding," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 199–207, Mar. 1981.

[11] D. W. Jones, "Application of splay trees to data compression," *Comm. ACM*, vol. 31, no. 8, pp. 996–1007, 1989.

[12] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop.*, vol. 23, no. 2, pp. 149–162, 1979.

[13] F. Rubin, "Arithmetic stream coding using fixed precision registers," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 672–675, Nov. 1979.

[14] B.Ya. Ryabko, "Data compression by means of a book stack, *Probl. Inform. Transm.*, vol. 16, no. 4, pp. 16-21, 1980 (in Russian). (In English: vol. 16, no. 4, pp. 265-269, 1981).
[15] ——, "Letter in *Commun. ACM*, vol. 30, no. 9, pp. 792, 1987.
[16] ——, "A fast sequenial code," *Soviet Math. Doklady*, vol. 39, no. 3, pp. 533-537, 1989.
[17] ——, "The fast on-line adaptive code," *Probl. Inform. Transm.* (to be published, in Russian).
[18] Yu.M. Starkov, "Shannon codes," *Probl. Inform. Transm.*, vol. 20, no. 9, pp. 3-16, 1984 (in Russian).
[19] J. A. Storer, *Data Compression. Method and Theory*. New York: Computer Science Press, 1988.
[20] J. S. Vitter, "Two papers on dynamic Huffman codes," Techn. Rep. CS95-13. Brown Univ., Dept. Comput. Sci., Providence, RI. Revised Dec. 1986.
[21] F. M. J. Willems, "Universal data compression and repetition times," *IEEE Trans. Inform. Theory*, vol. 35, pp. 54-58, Jan. 1989.
[22] J. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520-540, 1987.
[23] J. Ziv and A. Lempel, "Compression of individual sequences via variable rate coding," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 530-536, Sept. 1978.

The T_4 and G_4 Constructions for Costas Arrays

Solomon W. Golomb

***Abstract*—Two of the algebraic constructions for Costas arrays, designated as T_4 and G_4, are described in detail, and necessary and sufficient conditions are given for the sizes of Costas arrays for which these constructions occur. These constructions depend on the existence of primitive roots satisfying certain equations in finite fields.**

***Index Terms*—Costas arrays, primitive roots, Lempel's construction.**

In [1], a number of systematic algebraic constructions for Costas Arrays are described. The validity of several of these constructions is proved in [2]. However, two of the algebraic constructions, designated T_4 and G_4 in [1], are not discussed in [2]. The present note proves the assertions made in [1] concerning these two constructions, and furnishes some additional algebraic information.

We briefly review a few of the definitions from [1] and [2].

Definition 1: A *Costas array* of order n is an $n \times n$ permutation matrix with the property that the $\binom{n}{2}$ vectors connecting two 1's of the matrix are all distinct as vectors. (That is, no two vectors are equal in both magnitude and slope).

Specifically, if we have four distinct entries $a_{i_1 j_1} = a_{i_2 j_2} = a_{i_3 j_3} = a_{i_4 j_4} = 1$ in the matrix, we must not have $(i_2 - i_1, j_2 - j_1) = (i_4 - i_3, j_4 - j_3)$, nor may we have $(i_2 - i_1, j_2 - j_1) = (i_3 - i_2, j_3 - j_2)$.

Definition 2: If $n = q - 2$, where $q = p^k$ is the size of a finite field, then the *Lempel construction* L_2 for a Costas array of order n sets $a_{ij} = 1$ iff $\alpha^i + \alpha^j = 1, 1 \le i, j \le q - 2$, where α is any fixed primitive root in GF(q). (Note that the Lempel construction gives a *symmetric* permutation matrix with the Costas property.)

Definition 3: The T_4 construction occurs for GF(q) iff there is a primitive element α in GF(q) with $\alpha + \alpha^2 = 1$.

Manuscript received August 29, 1991. This work was supported in part by the United States Office of Naval Research, under Grant No. N00014-90-J-1341.

The author is with the Department of Electrical Engineering-Systems, University of Southern California, University Park, EEB-504a, Los Angeles, CA 90089-2565.

IEEE Log Number 9107518.

H. Taylor's T_4 construction leads to a Costas Array of order $n = q - 4$. Specifically, from the Lempel construction L_2, using the primitive root α which satisfies $\alpha + \alpha^2 = 1$ in GF(q), we have both $\alpha^1 + \alpha^2 = 1$ and $\alpha^2 + \alpha^1 = 1$. Thus both $a_{12} = 1$ and $a_{21} = 1$ in the L_2 construction of order $q - 2$. Removing the two topmost rows and the two leftmost columns from the L_2 array leaves a Costas array of order $q - 4$, which is the T_4 array.

Theorem 1: A necessary condition for the T_4 construction is that q is 4, or 5, or 9, or a prime p with $p \equiv \pm 1 \pmod{10}$.

Proof: We are asking for a field GF(q) in which the equation $x^2 + x - 1$ *has* roots, *and* in which at least one of these roots is primitive in GF(q).

Over GF(2), $x^2 + x - 1$ is irreducible, but generates GF(4). In no other field of characteristic 2 will a root of $x^2 + x - 1$ be primitive, since these roots have only primitivity 3.

Over GF(3), $x^2 + x - 1$ is irreducible, but generates GF(9). In no other field of characteristic 3 will a root of $x^2 + x - 1$ be primitive, since these roots have only primitivity 8.

For $p > 5$, the roots of $x^2 + x - 1 = 0$ are given by the quadratic formula as $(-1 \pm \sqrt{5})/2 \equiv (-1 \pm \sqrt{5})((p+1)/2) \pmod{p}$, which lie in GF($p$) iff 5 is a quadratic residue modulo p, and in GF(p^2) otherwise. Now, 5 is a quadratic residue modulo $p > 5$ iff $p \equiv \pm 1 \pmod{10}$, from the law of quadratic reciprocity:

$$\left(\frac{5}{p}\right) = \left(\frac{p}{5}\right)(-1)^{(p-1)/2 \cdot (5-1)/2} = \left(\frac{p}{5}\right), \quad \text{and} \quad \left(\frac{a}{5}\right) = +1 \text{ iff } a \equiv \pm 1 \pmod{5};$$

and since p is odd, this becomes $p \equiv \pm 1 \pmod{10}$. Thus, only primes with unit digit 1 or 9 are candidates for the T_4 construction, and the construction occurs iff at least one root of $x^2 + x - 1$ is primitive in GF(p). (It is possible for neither root, exactly one root, or both roots, to be primitive, depending on p.)

If 5 is a quadratic nonresidue modulo $p > 5$, then the roots of $x^2 + x - 1$ lie in GF(p^2) but not in GF(p). However, they cannot be primitive in GF(p^2), because a necessary condition for $x^2 + x + g$ to have primitive roots in GF(p^2) is that g be primitive in GF(p), and the only prime fields in which $g = -1$ is primitive are GF(2) and GF(3).

Finally, for $p = 5$, $\alpha = 2$ is a root of $x^2 + x - 1 \pmod{5}$, and the T_4 construction occurs in this case. (Here, in effect, $\sqrt{5} = 0$, and $x^2 + x - 1 = (x - 2)(x - 2)$ has a repeated root.) □

The following generalization G_2 of Lempel's construction is due to Golomb [2].

Definition 4: Let α and β be any two primitive roots in GF(q). Then G_2 is the Costas array of order $n = q - 2$ for which $a_{ij} = 1$ iff $\alpha^i + \beta^j = 1$.

Moreno *et al.* [3] have proved Golomb's Conjecture A in [2], which asserts: Every finite field GF(q) with $q > 2$ contains two primitive roots α and β (not necessarily distinct) with $\alpha + \beta = 1$. This leads to the following definition.

Definition 5: For every $q = p^k > 3$, the G_3 *construction* for a Costas array of order $n = q - 3$ is obtained from the G_2 construction using α and β with $\alpha + \beta = 1$ (which has $a_{11} = 1$, since $\alpha^1 + \beta^1 = 1$), by removing the topmost row and the leftmost column.

Definition 6: The G_4 *construction* occurs for GF(q) iff there are two primitive elements, α and β, in GF(q) with $\alpha + \beta = 1$ and $\alpha^2 + \beta^{-1} = 1$.

If both $\alpha + \beta = 1$ and $\alpha^2 + \beta^{-1} = 1$, the G_2 construction for a Costas array of order $n = q - 2$ has $a_{11} = 1$ (from $\alpha^1 + \beta^1 = 1$)