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, \( C(a) \Phi \) is \( E_i \oplus E_i \) where \( E_i \) is the \([2,3,1] \) tetracode (see [12]); but the number of possible codes for \( E_i \Phi \) is extremely large. However, 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 there are 11 inequivalent codes for values \( i = 1, 5, 19, 29, 49, 59, 65, 83, 97, 135, \) and 259. Here \( \mu^2 = 0201020101101001020 \), \( \mu^3 = 002112101201001020 \), \( \mu^4 = 0122101201001020 \), \( \mu^5 = 021002121102001020 \), \( \mu^6 = 21000212121011001020 \), \( \mu^7 = 22100022012112121 \), \( \mu^8 = 2212100021211221 \), \( \mu^9 = 211000120012210020 \), \( \mu^{10} = 002100212202200200 \), \( \mu^{11} = 2112100221211002021 \), and \( \mu^{12} = 02002002202020000 \).

### 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, \( C(a) \Phi \) is \( E_i \oplus E_i \) where \( E_i \) is the \([2,3,1] \) tetracode (see [12]); but the number of possible codes for \( E_i \Phi \) is extremely large. However, 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 there are 11 inequivalent codes for values \( i = 1, 5, 19, 29, 49, 59, 65, 83, 97, 135, \) and 259. Here \( \mu^2 = 0201020101101001020 \), \( \mu^3 = 002112101201001020 \), \( \mu^4 = 0122101201001020 \), \( \mu^5 = 021002121102001020 \), \( \mu^6 = 021002121102001020 \), \( \mu^7 = 21000212121011001020 \), \( \mu^8 = 22100022012112121 \), \( \mu^9 = 2212100021211221 \), \( \mu^{10} = 211000120012210020 \), \( \mu^{11} = 2112100221211002021 \), and \( \mu^{12} = 02002002202020000 \).

A Fast On-Line Adaptive Code

Boris Ya. Ryabko

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

**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, Sleator, Tarjan Manuscripts received January 17, 1990. The author is with Applied Mathematics and Cybernetics, Novosibirsk Institute of Communication, Kirov Street 86, Novosibirsk-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 \(x_1, x_2, \ldots, x_n \geq 1\), in a finite alphabet \(A = \{a_1, \ldots, a_k\}\), \(k \geq 2\). The code of a symbol \(x_i\), \(1 \leq i \leq N\), in on-line coding may depend on \(x_1, \ldots, x_{i-1}\) but should be independent of \(x_{i+1}, \ldots, x_N\). 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 in the storage capacity of the encoder and decoder (when they are implemented by a computer). The main objective will be to determine efficiency estimates asymptotic to the alphabet size, \(n\), the number of symbols in all possible words of the alphabet \(A\). Formally, 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 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 to the alphabet size, \(n\), the number of symbols in the alphabet. Since \(n\) is quite large (\(2^N \gg 2\)), 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 \(O(n)\). Therefore, for \(P(a) = 2^{-a}\), the codeword length for the symbol \(a\) will be equal to \(n\) bits, and the encoding and decoding time, thus equal to \(O(n)\) (because it is necessary to scan a codeword both while encoding and decoding). Looking ahead, we note that the maximum encoding and decoding time is only possible when using a code having a maximum codeword length that does not much exceed \(n\).

For the “book stack” method, the redundancy, when the number of symbols, \(n\), in the alphabet is large, equals \(\log n + O(1)\). In [2], a simple implementation of “book stack” is proposed when the maximum time of encoding and decoding is \(O(\log n)\). Note that this value is fairly close to the obvious lower bound, \(O(\log n)\). 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 \(n + O(1)\) 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 \(O(n \log n)\) bits for \(n \to \infty\). Some have relatively large redundancy, \(\log n + O(1)\), and asymptotically small time required for encoding and decoding, \(O(\log^2 n)\).

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

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 aforementioned 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 \(\omega\), is designed for coding words \(x_1, x_2, \ldots, x_N\) of alphabet \(A = \{a_1, \ldots, a_k\}\). 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 \(x_{i-w+1}, \ldots, x_i\), 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 code is repeated. The main problem is associated with performing the encoding and decoding operations in the time \(O(\log^2 n)\).

Let

\[
I = \left\lfloor \log n \right\rfloor, \quad L = 2^I.
\]

During the coding according to method \(\omega\), we will consider only the window of length \(w\) for

\[
w = \left(2^r - 1\right)L, \quad (2)
\]

where \(r\) is an integer. For a description of \(\omega\), it is convenient to extend a word \(x_1, x_2, \ldots, x_N\) by \(w\) symbols to the left as \(\ldots, a_3a_2a_1a_1a_1a_1x_1x_2\ldots, x_N\). The word obtained will be expressed as \(x_{1-w+1}, \ldots, x_{1+w}, x_1, x_2, \ldots, x_N\). (This allows introduction of the window for encoding the first symbols \(x_1, x_2, \ldots,\) 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 \(x_1, x_2, x_3, \ldots, x_i\). Define \(\tilde{P}\), as

\[
\tilde{P}(a, j, i) = P(a, j, i) + 1; \quad i = 1, \ldots, n; \quad 1 \leq i \leq N;
\]

\[
\tilde{P}(a, j, i) = 1, \quad \text{for } j = n + 1, \ldots, L
\]

Then

\[
Q(a, i) = 2 \sum_{k=1}^{i-1} \tilde{P}(a, i) + P(a, i) >\]

\[
m(a, j) = l + r + 1 - \left\lfloor \log \tilde{P}(a, j) \right\rfloor; \quad j = 1, \ldots, n;
\]

\[
i = 1, \ldots, N
\]

\(l\) 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 \(m(x_i, l)\) symbols from the binary expansion of \(Q(x_i, l)\). Thus, the
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 \(A = \{a_1, \ldots, a_n\}\) with the set of probabilities \((P(a_1, i) + 1)/(2L), j = 1, \ldots, n\). From the definition of \(P(a_1, i)\) 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 fastest execution of encoding and decoding.

To begin with, we will describe the encoding. When encoding a symbol \(x_i\) from a word \(x_1, x_2, \ldots, x_n \in A^*\), it is necessary 1) to calculate the \(P(x_i, i)\)'s that are given in (4) in order to form the codeword based on them, and 2) alter the frequencies \(P(a_i, i + 1), a \in A\), because of the window shift, according to the formulae:
\[
\begin{align*}
\hat{P}(x_i, i + 1) &= \hat{P}(x_i, i) + 1, \\
\hat{P}(x_{i-L}, i + 1) &= \hat{P}(x_{i-L}, i) - 1, \\
\hat{P}(a_i, i + 1) &= \hat{P}(a_i),
\end{align*}
\]

for \(a \neq x_i, \quad a \neq x_{i-L} - 1\). (5)

To perform these operations quickly, a computer memory has to store \((2L - 1)\) numbers defined by the equalities:
\[
D^m_{j}(i) = \hat{P}(a_j, i), \quad j = 1, \ldots, L; \quad i = 1, \ldots, N \quad (6)
\]

(for \(j = n + 1, \ldots, L\), defined \(\hat{P}(a_j, i) = 1\), and
\[
D^{m+1}_{j}(i) = D^m_{j}(2j - 1) + D^m_{j}(2j), \quad j = 1, \ldots, 2^{L-m} + 1 \quad (7)
\]

We now show how to compute \(D^m_{i}(j), \hat{P}(a_i, i)\), for any \(k = 1, \ldots, n\), using a set \(D^m_{j}(i); m = 1, \ldots, L; j = 1, \ldots, 2^{L-m} + 1; i = 1, \ldots, N\), in \(O(\log^2 n)\) operations with bits. Let the binary expansion of \(k\) be
\[
k = \tau_s 2^{s-1} + \tau_{s-1} 2^{s-2} + \cdots + \tau_1 2^0,
\]

where \(\tau_s = 0\) or 1 for \(s = 1, 2, \ldots, 1\). Define
\[
t_m = \sum_{s=0}^{L-m} \tau_s 2^{l-m-s}, \quad m = 1, \ldots, L. \quad (9)
\]

Then, the following equality follows from (6)-(9):
\[
\sum_{m=1}^{k} \hat{P}(a_m, i) = \sum_{t=1}^{t_m} \tau_s D^m_{t}(s). \quad (10)
\]

The right sum in (10) consists of no more than one 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 \(\{D^m_{j}(i)\}\) 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 \(\sum^k_{j=1} P(a_j, i) = w\). From (3), it follows that \(\sum^k_{j=1} P(a_j, i) = w + L\). Since \(w + L = 2L\) (see (2)), then to form a codeword according to (4), it is necessary to sum no more than one number that does not exceed \(2L\). Hence, no more than \(l(l + 1)\) operations with one-bit words are required. To form \(t_m\) according to (9), we need \(l(l + 1)\) operations with one-bit words. In addition to forming a codeword by (4), it is necessary to alter the frequencies \(P(a, i)\) according to (5) and, respectively, alter the sets \(\{D^m_{j}(i)\}\), in (6), (7).

Before the description of the decoding, consider an example. Assuming \(A = \{a_1, a_2, \ldots, a_n\}\), we want to encode the word \(x_1, x_2, \ldots, x_10 = a_6a_6a_6a_6a_6a_6a_6a_6a_6a_6\). Here, \(L = 8, l = 3\). Assume the window width \(w = 8\), i.e. \(w = L\), \(r = 1\), see (2). Consider the coding of symbol \(x_{10}\). There are one \(a_6\) and seven \(a_6\) in the word, so \(P(a_6, 10) = P(a_6, 10) = \cdots = P(a_6, 10) = P(x_{10}) = 1\), \(P(x_{10}, 10) = 7, P(a_6, 10) = 0\). According to (3) and (6), (7), we have \(P(a_6, 10) = P(a_6, 10) = \cdots = P(a_6, 10) = P(a_6, 10) = 1\), \(P(a_6, 10) = 8, P(a_6, 10) = 2\); \(D^0_{j}(1) = D^0_{j}(2) = \cdots = D^0_{j}(8) = 1, D^0_{j}(6) = 8, D^0_{j}(7) = 2;\)

\(D^1_{j}(1) = D^1_{j}(2) = 2; D^1_{j}(3) = 9; D^1_{j}(4) = 3; D^1_{j}(1) = 4; D^1_{j}(2) = 12\). To form the codeword of symbol \(x_{10}\), we need to find \(Q(a_6, 10)\), see (4), which, in turn, requires computing \(\sum_{m=1}^{k} \hat{P}(a_m, 10)\). From (10),
\[
\sum_{m=1}^{k} \hat{P}(a_m, 10) = \sum_{i=1}^{t_m} \tau_i D^m_{t}(s). \quad (11)
\]

For \(k = 5\), (8) and (9) yield \(\tau_1 = 1, \tau_2 = 0, \tau_3 = 1,\)
\[
t_1 = \tau_1 2^1 + \tau_2 2^1 + \tau_3 2^0 = 5, \tau_2 = \tau_2 2^1 = 2, \tau_3 = \tau_3 2^0 = 1.
\]

From this and (11), we obtain
\[
\sum_{m=1}^{5} \hat{P}(a_6, 10) = 1 \cdot D^0_{j}(5) + 0 \cdot D^0_{j}(2) + 1 \cdot D^1_{j}(1) = 1 \cdot 1 + 1 \cdot 4 = 5.
\]

Then, using one multiplication by 2 and one addition operation, we have
\[
Q(a_6, 10) = 25 + 8 = 18 = 1 \cdot 2^4
\]

\[
+ 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0
\]

\[
m(a_6, 10) = 1 + 1 + 3 = \log \hat{P}(a_6, 10) = 2.
\]

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

Now we shall describe the decoding procedure. Assume that \(x_1, x_2, \ldots, x_n\) was encoded using the frequency code as \(\omega(x_1 / x_2 / \cdots / x_N) \cdots \omega(x_N / x_1 \cdots x_{n-1})\), the symbols \(x_1, x_2, \ldots, x_N\) being already decoded based on the prefix \(\omega(x_1) \cdots \omega(x_{n-1})\).
After that, a number $\omega(x_i,\ldots x_{i+N})$, and it is necessary to decode $x_i$, $i = 1, 2,\ldots, N$. Both in encoding and decoding, the values of $D^m(j); m = 1, \ldots, i; j = 1,\ldots, 2^{1-m+1}$ are formed. In order to find $x_i$, the decoder reads out $l + r + 1$ symbols of the word $\omega(x_i,\ldots x_{i-1}) \cdot \omega(x_{N+1},\ldots x_{N+i})$. These binary characters can be denoted as $u_{i,2} \cdots u_{i,r+1}$. Using them, the following number is formed
\begin{equation}
\sigma + 2^{r+1}u_{i,1} + 2^{r+1}u_{i,2} + \cdots + 2^ru_{i,r+1}.
\end{equation}

After that, a number $m$ is found, such that
\begin{equation}
2 \sum_{j=1}^{m-1} \tilde{P}(a, j) \leq u < 2 \sum_{j=1}^{m} \tilde{P}(a, j).
\end{equation}

Then the $i$th symbol in the encoded word $x_i,\ldots x_N$ has to be $a_m$, i.e., $x_i = a_m$. (This follows from (4): the difference between $u$ and $Q(a, m)$ is not more than $P(a, m)$.)

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

Proceeding similarly, we can see that in one step a number, $m$ is 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 $2^r \cdot 4$ and so the total number of operations with one-bit words is proportional to $l(l + r)$. After that, the decoder defines the codeword $\omega(x_i,\ldots x_N)$, i.e., $x_i,\ldots x_{i+N}$ the same way as in encoding. Then, it becomes clear which characters in $u_{i,2} \cdots u_{i,r+1}$ constitute $\omega(x_i,\ldots x_{i+N})$ and which belong to $\omega(x_{i+1},\ldots x_{i+N})$. The procedure for computing $u$ according to (12) requires $(l + r + 1)$-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 $x_i$ in the word $a_8a_9a_1a_2a_3a_4a_5a_6a_7$.* The decoder reads out $l + r + 1$-bit words of the bit $\omega(x_i,\ldots x_{i+N})$. In order to decode $x_i$, we use the following statement. Then, $u$ is computed (see (12)) as
\begin{equation}
\sigma + 2^{r+1} + 2^{r} \cdot 0 + 2^{2} \cdot 1 + 2^{3} \cdot 0 + 2^{5} \cdot 1 + 2^{2} \cdot 0 + 2^{4} \cdot 0.
\end{equation}

Obviously, for any $V_iV_jV_k$, $16 \leq u \leq 23$. Further, $u$ and $2D^m(1) = 2 = 4$ are compared. Since $u > 8$, then $m$ satisfies $4 < m \leq 8$. Further, $2D^m(1) + D^m(3) = 2(4 + 1) = 10$ is computed. Since $u > 26$, then $4 < m \leq 6$. After that, $2D^m(1) + D^m(3) = 2(4 + 1) = 10$ is computed. Since $u > 10$, then $5 < m \leq 6$, i.e., $m = 6$, and, consequently, $x_i = a_6$.

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 $X_{i-1} \cdots X_{i-1}$, each requiring 1 bits for recording. It is necessary to store numbers $\{D^m(j); m = 1, \ldots, i; j = 1, \ldots, 2^{1-m+1}\}$ (2L - 2 numbers in total), each being stated as a word of $(l + r + 1)$ bits. Thus, $w + 2L(l + r)$ bits are needed for the storage of these numbers. From this and (2), it follows that the total storage capacity is proportional to $2L \cdot l$ bits. So the following statement can be formulated concerning the complexity of a frequency code.

**Statement:** Let $A = \{a_1, \ldots, a_m\}$, $n \geq 2$, be an alphabet, $x_1,\ldots x_N \in A^N$, $N \geq 1$, and suppose that the word $x_1,\ldots x_N$ is encoded by the frequency code having the window length $w = (2^q - 1)2^{1[\log_n r]}$ 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 $2n \cdot \log n$.

**III. The Redundancy of the Frequency Code**

Let $x = x_1, x_2, \ldots x_N$ be a word in the alphabet $A$ and let $\varphi$ be an on-line code. Let
\begin{equation}
\epsilon(\varphi, x) = N^{-1} \sum_{i=1}^{N} \epsilon_{a, i} = N^{-1} \sum_{i=1}^{N} \mu(u) c(\varphi, u),
\end{equation}

where $A^N$ is the set of words of length $N$ in the alphabet $A$, and $\mu(x_1, x_2, \ldots x_N) = \mu(x_1) \mu(x_2) \cdots \mu(x_N)$. Denote as $\tilde{e}(\varphi)$ the average codeword length of Huffman code constructed for the alphabet $A$ with probability distribution $\mu$. Define
\begin{equation}
\hat{e}(\varphi, \mu) = \epsilon(\varphi, \mu) - \tilde{e}(\mu),
\end{equation}

where the supremum is taken over all Bernoulli sources.

The next theorem characterizes the properties of the code $\omega$.

**Theorem:** Let $A = \{a_1,\ldots, a_m\}$ be an alphabet and $\omega$ the frequency code having the window $(2^q - 1)2^{1[\log n]}$. For any Bernoulli source generating symbols from $A$, the redundancy does not exceed $2 + \log e/(2^q - 1)$.

The proof is provided in [17]. The analogous result is also correct for redundancy defined for individual sequences (117).

**References**


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 $a_{i,j} = a_{k,l} = a_{l,j} = a_{i,k} = 1$ 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 $(i_2 - i_1, j_2 - j_1) = (i_4 - i_3, j_4 - j_3)$.

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 $a_{i,j} = 1$ if $\alpha i + \beta j = 1, 1 \leq i, j \leq q - 2$, where $\alpha$ 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 $GF(q)$ if there is a primitive element $\alpha$ in $GF(q)$ with $\alpha + \alpha^2 = 1$.

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 $\alpha$ which satisfies $\alpha + \alpha^2 = 1$ in $GF(q)$, we have both $\alpha^3 + \alpha = 1$ and $\alpha^3 + \alpha^2 = 1$. Thus both $a_{i,j} = 1$ and $a_{i,j} = 1$ 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 $p = \pm 1 \pmod{10}$.

Proof: We are asking for a field $GF(q)$ in which the equation $x^2 + x - 1 = 0$ 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 $(-1 \pm \sqrt{5})/2 = (-1 \pm \sqrt{5})/(p + 1)/2$ (mod $p$), which lie in $GF(p)$ if 5 is a quadratic residue modulo $p$, and in $GF(p^2)$ otherwise. Now, 5 is a quadratic residue modulo $p > 5$ if $p = \pm 1 \pmod{5}$, from the law of quadratic reciprocity:

$$\left(\frac{5}{p}\right) = \left(\frac{p}{5}\right) = (-1)^{(p-1)/2}(-1)^{(5-1)/2} = \left(\frac{p}{5}\right),$$

and $\left(\frac{5}{p}\right) = 1$ if $a = \pm 1 \pmod{5}$.

Since $p$ is odd, this becomes $p = \pm 1 \pmod{10}$. 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 residue 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 $x^2 + x - 1$ (mod 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 $G_4$ of Lempel's construction is due to Golomb [2].

Definition 4: Let $\alpha$ and $\beta$ be any two primitive roots in $GF(q)$. Then $G_4$ is the Costas array of order $n = q - 2$ for which $a_{i,j} = 1$ if $\alpha i + \beta j = 1$.

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 $\alpha$ and $\beta$ (not necessarily distinct) with $\alpha + \beta = 1$. This leads to the following definition.

Definition 5: For every $q = p^k > 3$, the $G_4$ construction for a Costas array of order $n = q - 3$ is obtained from the $G_3$ construction using $\alpha$ and $\beta$ with $\alpha + \beta = 1$ (which has $a_{i,j} = 1$, since $\alpha^2 + \beta = 1$), by removing the topmost row and the leftmost column.

Definition 6: The $G_4$ construction occurs for $GF(q)$ if there are two primitive elements $\alpha$ and $\beta$, in $GF(q)$ with $\alpha + \beta = 1$ and $\alpha^2 + \beta = 1$.

If both $\alpha + \beta = 1$ and $\alpha^2 + \beta = 1$, the $G_4$ construction for a Costas array of order $n = q - 2$ has $a_{i,j} = 1$ (from $\alpha^2 + \beta = 1$)