An Implementation of Elias Coding for Input-Restricted Channels

ERDAL ARIKAN, MEMBER, IEEE

Abstract — **An implementation of Elias coding for input-restricted channels is presented and analyzed. This is a variable-to-fixed length coding method that uses finite-precision arithmetic and can work at rates arbitrarily close to channel capacity as the precision is increased. The method offers a favorable tradeoff between complexity and coding efficiency. For example, in experiments with the [2, 7] runlength constrained channel, a coding efficiency of 0.9977 is observed, which is significantly better than what is achievable by other known methods of comparable complexity.**

I. Introduction

Ever since Shannon [1, p. 36] formulated and studied input-restricted channels, many coding methods have been proposed for such channels; the reference section provides a partial list. We present another method, based on Elias coding, which may be of interest since it can achieve extremely high coding efficiencies using little memory and computation.

The idea of using Elias coding for input-restricted channels is not new, though the earlier codes were presented under the name of arithmetic coding [4], [5], [6]. The primary interest in [4], [5] was in fixed-rate implementations. Here, we consider a variable-to-fixed-length implementation that is fixed-rate only in a certain asymptotic sense. It return for giving up the strict fixed-rate property, a favorable tradeoff is obtained between coding complexity and efficiency, as demonstrated by experimental and analytical results.

The relation between Elias coding and the code here is a special case of the duality between source coding (data compaction) and coding for input-constrained channels (data translation). The decoder for a data-translation code is an encoder for a data-compaction code: it removes the structural redundancy that is present in the constrained channel sequence. This relationship was stated and exploited in [4], [5], and more recently in [16]. For the code here, the decoder is an Elias encoder, more exactly a finite-precision floating-point implementation of Elias coding, similar in some respects to that in [15]. (For Elias coding and its variants, see [14].)

The main results of this correspondence are contained in Section IV. Sections II and III contain definitions, conventions, and some background results. Section V contains complementary remarks on the algorithm.

II. Conventions and Definitions

We consider only input-restricted channels that can be presented by finite state diagrams. An example of such a channel is the [2, 7] runlength constrained channel, presented by the state diagram in Fig. 1. (Runlength constraints arise, e.g., in magnetic recording applications. For details, see, e.g., [13].) Each edge in Fig. 1 is labeled by a 0 or a 1, the input letters for this channel. A 0–1 sequence is admissible as an input to the [2, 7] channel iff it can be generated by a walk through the state diagram. It will be noted that runlengths of 0 in admissible sequences are bounded between 2 and 7, and being in state i implies that the current runlength of 0 is $i-1$.

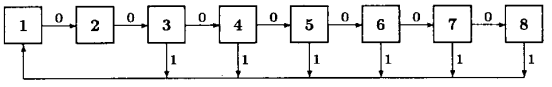

Fig. 1. State diagram for [2, 7] runlength constraints.

There may be several reasonable ways of presenting an input-restricted channel by a state diagram. For our purposes any such presentation is acceptable as long as it has the properties of being simple and deterministic. A state diagram is said to be simple if for each pair of (not necessarily distinct) states i and j, there is at most one edge from state i to state j. A state diagram is said to be deterministic if, for each state, the labels on edges emerging from that state are distinct.

Note that, if one is given a sequence of edge-labels through a deterministic state diagram and the initial state is known, one can then determine the sequence of states visited by the walk. (This notion was expressed by the terms unifilar in [2] and, in a somewhat weaker form, by right-resolving in [9].)

It is easy to see that if an input-restricted channel can be presented by a finite state diagram, so can it be by a simple, deterministic one. So, the restrictions we have imposed on state diagram presentations do not exclude any channels. We shall, however, exclude from consideration those channels that cannot be presented by irreducible state diagrams. A state diagram is said to be irreducible if there exists a directed path from each state to each other state. Channels that are not irreducible are not of particular interest; they are best studied in terms of their irreducible components. Henceforth, we shall assume that the state diagram under consideration is finite, simple, deterministic, and irreducible.

Let us note that the restriction to simple state diagrams is made only to simplify the notation; the coding method presented here can be applied to nonsimple state diagrams after minor changes. The restriction to deterministic state diagrams is essential for decodability, though that too can be relaxed somewhat (to what is called right-resolving in [9]).

We shall label the states by integers 1 through S, where S denotes the number of states. We shall let $t_{i,j}$, $1 \le i,\ j \le S$, denote the number of edges from state i to state j. Thus, $t_{i,j}$ will be either 0 or 1 since we consider only simple state diagrams. The matrix $T = (t_{i,j})$ will be called the state transition matrix. We shall let $l_{i,j}$ denote the channel symbol labeling the edge from state i to state j; if there is no such edge, $l_{i,j}$ will be undefined.

III. Maxentropic Probabilities and Channel Capacity

Let λ be the largest positive eigenvalue of T, the state-transition matrix for some input-restricted channel. Let $(B_i,\ 1 \le i \le S)$ be an eigenvector belonging to λ: $TB = \lambda B$. (By the Frobenius theorem for irreducible, nonnegative matrices [3, p. 53], B is unique up to a scaling constant, and one may assume that $B_i > 0$ for all i.)

Shannon [1, p. 58] showed that the entropy of the input to an input-restricted channel is maximized by a source that visits the states of the constraint diagram according to the following Markovian transition probabilities:

$$p_{i,j} = t_{i,j} B_j \lambda^{-1} / B_i. \tag{1}$$

Manuscript received December 7, 1987; revised May 13, 1989. This research was supported in part by Joint Services Electronics Program under contract N00014-84-C-0149. This correspondence was presented in part at the IEEE International Symposium on Information Theory, Kobe, Japan, June 1988.

The author is with the Department of Electrical Engineering, Bilkent University, P.K. 8, 06572, Maltepe, Ankara, Turkey.

IEEE Log Number 8933104.

These transition probabilities are called maxentropic, and the resulting maximum entropy is calculated to be $\log \lambda$.

Shannon [1, p. 37] also showed that the capacity of an input-restricted channel is given by $C = \log \lambda$. Thus, in order to use this noiseless channel at the maximum possible rate C, one must maximize the entropy of the channel input. This suggests that the output of an encoder working at a rate close to channel capacity should, in some sense, approximate the maxentropic Markov chain. The algorithm presented in the next section tries to achieve this as best it can under the limitations of finite-precision arithmetic.

IV. The Algorithm

The algorithm presented here is a variable-to-fixed-length sliding-block code, which in each encoding cycle accepts a variable number t of new data bits into the encoder and sends out one channel symbol. The encoder thus maps a data sequence $d_1, \cdots, d_L$ to a code sequence $x_1, \cdots, x_N$. The decoder processes the code sequence $x_1, \cdots, x_N$ and generates $d_1, \cdots, d_{L-K}$, the original data sequence except for a tail of K bits. The number K of undecodable bits is data-dependent, but always upperbounded by r, the length of registers used by the algorithm. Thus, by appending to the original data sequence a fixed tail of r bits, one can ensure the decodability of all data bits. Since we shall be primarily interested in the asymptotic performance of the algorithm as L goes to infinity, we shall ignore these end-effects.

In order to apply the algorithm to an input-restricted channel, we first fix a state-diagram presentation. Then, we choose a matrix of transition probabilities $(p_{i,j})$ on this state-diagram so that $p_{i,j} = 0$ whenever $t_{i,j} = 0$, and $\sum_{j=1}^{S} p_{i,j} = 1$, all i. The algorithm can work with any such set of transition probabilities, but what is intended is to choose the $(p_{i,j})$ as close to the maxentropic probabilities as possible within finite precision. It is aimed to have the encoder behave like a Markov chain (in its transitions from one state to another during the encoding process) with the chosen transition probabilities when the encoder input is a symmetric Bernoulli process (independent, equiprobable bits).

A pseudocode for the algorithm is shown next.

```
procedure encode((F_{i,j}), (l_{i,j}), r, s_0);
begin
begin{initialization}
s := s_0; y := 0; z := 2^r - 1; w := z - y;
u := Σ_{i=1}^{r} d_i 2^{r-i}
end{initialization}
while not end-of-data do
begin{encodingcycle}
determine the least i such that
F_{s,i} - F_{s,i-1} > 0 and y + ⌊wF_{s,i}⌋ ≥ u;
output the next code letter x = l_{s,i};
z := y + ⌊wF_{s,i}⌋;
y := y + ⌊wF_{s,i-1}⌋;
t := t(y, z);
shift t 0's into register y;
shift t 1's into register z;
shift the next t data bits into register u;
w := z - y; s := i
end{encodingcycle}
end.

procedure decode((F_{i,j}), (l_{i,j}), r, s_0);
begin
begin{initialization}
s := s_0; y := 0; z := 2^r - 1; w := z - y
end{initialization}
while not end-of-code-sequence do
begin{decodingcycle}
input the next code letter x;
determine the state i such that l_{s,i} = x;
z := y + ⌊wF_{s,i}⌋;
y := y + ⌊wF_{s,i-1}⌋;
t := t(y, z);
output the leftmost t bits of register y;
shift t 0's into register y;
shift t 1's into register z;
w := z - y; s := i
end{decodingcycle}
end.
```

We shall first explain some variables and operations that appear in this pseudocode. For each pair of states, there is a cumulative probability defined by

$$F_{i,j} = \sum_{1 \le s \le j} p_{i,s}. \tag{2}$$

By convention, we take $F_{i,0} = 0$ for all i.

The variables u, y, and z represent r-bit, right-to-left shift-registers. Register u holds a window of r data bits: $u = d_{h+1}, \cdots, d_{h-r}$, where h is the number of bits that have been processed by the encoder until then, and left the system. The contents of register u are also treated as the integer $u = \sum_{i=1}^{r} d_{h-i} 2^{r-i}$. In the same manner, registers y and z are treated as both bit sequences and integers. Thus, setting $z = 2^r - 1$ is equivalent to loading register z with r 1's. Likewise, setting $u = \sum_{i=1}^{r} d_i 2^{r-i}$ loads register u with the first r data bits.

The function $t(y, z)$ gives the length of the common (left) prefix of registers y and z (e.g., if $y = 001001$ and $z = 001110$ (with $r = 6$), the common prefix is 001 and $t(y, z) = 3$).

The shift operation on registers is best explained by an example: If $u = 001010$ ($r = 6$), then executing "shift 111 into register u" results in $u = 010111$.

To explain how the algorithm works, let us indicate the value of a variable at the end of encoding cycle n by the subscript n (e.g., y_n, s_n, etc.). Let the subscript 0 indicate initial values. The algorithm works by generating, in effect, a sequence of intervals $I_n = [y_n, z_n]$, $n \ge 0$, with the property that $y_n \le u_n \le z_n$. In the nth encoding cycle, the initial interval $I_{n-1} = [y_{n-1}, z_{n-1}]$ is divided into subintervals so that there is one subinterval for each state. The subinterval for state i is empty if

$$p_{s_{n-1},i} = F_{s_{n-1},i} - F_{s_{n-1},i-1} = 0; \qquad \text{otherwise,}$$

it equals

$$\left[y_{n-1} + \left\lceil w_{n-1} F_{s_{n-1},i-1} \right\rceil,\ y_{n-1} + \left\lfloor w_{n-1} F_{s_{n-1},i} \right\rfloor \right]. \tag{3}$$

What is aimed here is to allocate a fraction $p_{s_{n-1},i}$ of the initial interval to state i. These subintervals may overlap at their endpoints, but what matters is that they cover all integers in I_{n-1}. (With a more complicated rule, one could guarantee disjoint subintervals and slightly improve the coding efficiency.) The encoder sets s_n equal to the least i such that u_{n-1} lies in the subinterval for state i. The symbol $x_n = l_{s_{n-1},s_n}$ is then sent to the channel.

Let us now look at the situation from the decoder's point of view. Suppose that the decoder has kept up with the encoder and

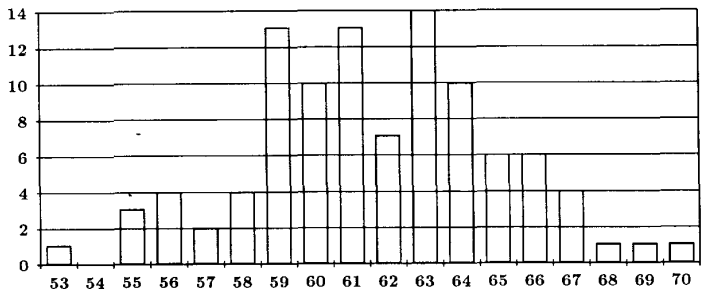

Fig. 2. Histogram of number of decoded digits minus 5100 for algorithm.

is in possession of y_{n-1}, z_{n-1}, and s_{n-1}. After observing x_n, the decoder can determine s_n (since the state diagram is deterministic), and hence, y_n, z_n, and the leftmost

$$t_n = t\left(y_{n-1} + \left\lceil w_{n-1} F_{s_{n-1}, s_n - 1} \right\rceil,\ y_{n-1} + \left\lfloor w_{n-1} F_{s_{n-1}, s_n} \right\rfloor\right) \quad (4)$$

bits of u_{n-1}. This explains why the encoder can drop, as it does, the leftmost t_n bits of register u, thus making room for the upcoming t_n data bits. The updating rule for registers y and z ensures that the inequality $y_n \le u_n \le z_n$ is satisfied going into the next cycle; so the process can continue indefinitely.

Thus the number of data bits that the decoder is able to decode after observing the first n code letters is given by $T_n = t_1 + \cdots + t_n$. The question that arises is whether the code here is asymptotically fixed-rate, i.e., whether $R_n = T_n/n$ converges as n goes to infinity to a fixed rate R for all data sequences. (In such limits, we assume that L, the number of data bits, is infinite.) Unfortunately, this is not so. However we shall see later in this section that, if each data sequence of a fixed finite length is assumed equally likely, then the set of all infinite-length data sequences for which the previous statement is false has total probability zero.

A. An Experimental Result

We applied the algorithm to the [2, 7] runlength constrained channel presented in Fig. 1. The numbers $(F_{i,j})$ were based on the following transition probabilities: $p_{3,1} = 0.34$, $p_{4,1} = 0.36$, $p_{5,1} = 0.40$, $p_{6,1} = 0.46$, $p_{7,1} = 0.59$, which were obtained by rounding off the maxentropic transition probabilities. The register length was $r = 8$.

We carried out the experiment 100 times. In each run the input to the encoder was a simulated symmetric Bernoulli sequence. The encoder was run until it produced $N = 10000$ code letters, which were then processed by the decoder, and the number T_N of decoded bits was recorded. Fig. 2 shows the frequency of occurrence of the observed values of T_N.

We observe that the sample mean of the rate $R_N = T_N/N$ equals 0.516165, which is within 0.23% of the channel capacity $C = 0.517370\cdots$ for the [2, 7] channel (see [13] for the capacity). We also observe that the spread of the observed values of T_N around the sample mean is, relatively speaking, quite small, suggesting some statistical regularity. The analysis that follows helps to explain these observations.

B. Statistical Analysis of the Algorithm

For the statistical analysis, we assume that the data sequence is a symmetric Bernoulli process. We first sketch a proof that R_N converges almost surely as N goes to infinity.

The proof is based on the observation that the sequence $(\sigma_n\colon n \ge 0)$, where $\sigma_n = (s_n, y_n, z_n)$, is a Markov chain with time-invariant transition probabilities. This follows from the facts that σ_n is a function of σ_{n-1} and u_{n-1}, and that u_{n-1} is uniformly distributed (equally likely to take on any integer value) on the interval $[y_{n-1}, z_{n-1}]$, regardless of the path taken to reach the state σ_{n-1}. (These statements can be proved easily by induction.)

Since the random variable t_n is a function of σ_{n-1} and σ_n, i.e., $t_n = t(\sigma_{n-1}, \sigma_n)$, we can write

$$R_N = 1/N \sum_{1 \le n \le N} t_n = \sum_{(\sigma, \sigma')} f_N(\sigma, \sigma') t(\sigma, \sigma') \quad (5)$$

where the random variable $f_N(\sigma, \sigma')$ equals $1/N$ times the number of transitions from state σ to state σ' during the first N transitions in the sequence (σ_n). Since (σ_n) is a Markov chain with finitely many states, the times at which a transition occurs from state σ to state σ' form a renewal process; hence, we have, almost surely as N goes to infinity (e.g., [17, p. 290]),

$$f_N(\sigma, \sigma') \xrightarrow{\text{a.s.}} \pi_\sigma r_{\sigma, \sigma'} \quad (6)$$

where π_σ is the reciprocal of the mean recurrence time of state σ, and $r_{\sigma,\sigma'}$ is the transition probability from state σ to state σ'. From (5) and (6), we have, as N goes to infinity,

$$R_N \xrightarrow{\text{a.s.}} R = \sum_{(\sigma, \sigma')} \pi_\sigma r_{\sigma, \sigma'} t(\sigma, \sigma'). \quad (7)$$

We have thus shown that the coding rate asymptotically approaches a constant R for almost all data sequences.

Next, we present an analysis, though inexact, provides insight into why and under what conditions we may expect R to be close to the channel capacity. We begin by noting the following recursive relation:

$$\begin{aligned} w_n &= z_n - y_n \\ &= \left(\left\lfloor w_{n-1} F_{s_{n-1}, s_n} \right\rfloor - \left\lceil w_{n-1} F_{s_{n-1}, s_n - 1} \right\rceil\right) 2^{t_n} + 2^{t_n} - 1. \end{aligned} \quad (8)$$

The algorithm aims at having

$$w_n \approx w_{n-1} p_{s_{n-1}, s_n} 2^{t_n} \quad (9)$$

within the limitations of finite-precision arithmetic. The idea is to have, as a consequence of the preceding equation,

$$w_N \approx w_0 p_{s_0, s_1} p_{s_1, s_2} \cdots p_{s_{N-1}, s_N} 2^{T_N}. \quad (10)$$

For $(p_{i,j})$ approximately maxentropic, this implies

$$w_N \approx w_0 \left(B_{s_N}/B_{s_0}\right) 2^{-NC + T_N}, \quad (11)$$

where C is the channel capacity. Since $1 \le w_N \le 2^r - 1$, it follows that, for large N,

$$R_N = T_N/N \approx C. \quad (12)$$

Thus to the extent that the previous approximations are valid for the state sequence (s_n) corresponding to a data sequence, the coding rate for that data sequence will be close to the channel capacity. The most effective way of ensuring the validity of these approximations for a large fraction of data sequences is to choose r, the register size, suitably large.

Note that in the limit of infinite precision, the above approximations are exact and the code is truly fixed-rate. The algorithm deviates from being fixed-rate as the precision is decreased. The experimental values of R_N previously reported suggest that even with a modest precision that deviation may be kept at an insignificant level.

C. The Error Propagation Problem

Suppose that, instead of the entire code sequence $x_1, x_2, \cdots$, the decoder is given $x_{n+1}, x_{n+2}, \cdots$. Can the data sequence still be reconstructed correctly allowing a finite number of initial errors? This situation arises when some of the first n symbols are garbled by noise, or when one wishes to start decoding in the middle of an encoded sequence. In such cases, one would not like to see a catastrophic propagation of decoding errors.

Unfortunately the codes generated by the algorithm may suffer catastrophic error propagation. We now examine this problem in more detail, mention possible remedies, but propose no specific solutions.

Let σ_n denote, as before, the state of the encoder at the end of encoding cycle n. Let $\hat{\sigma}_n$ denote the state of the decoder at the end of decoding cycle n. We may view $\hat{\sigma}_n$ as the decoders estimate of σ_n. Assume that $x_{n+1}, x_{n+2}, \cdots$ is error-free.

From our discussion of the decoding algorithm, it is clear that the output of the decoder at decoding cycles $> n$ is completely determined by $(\hat{\sigma}_n, x_{n+1}, x_{n+2}, \cdots)$. In other words the effect of $x_1, \cdots, x_n$ on the future behavior of the decoder is completely summarized by $\hat{\sigma}_n$. Thus for each possible value of $\hat{\sigma}_n$ (there are finitely many) there is a hypothetical data sequence. One of these sequences, the one for which $\hat{\sigma}_n = \sigma_n$, is the true data sequence. In order to stop error propagation, the decoder must be able to select one of the hypothetical sequences that shares a common suffix with the true sequence. (Two sequences $\alpha_1, \alpha_2, \cdots$ and $\beta_1, \beta_2, \cdots$ are said to share a common suffix if there exist k, k' such that $\alpha_{k+i} = \beta_{k'+i}$ for all $i \geq 1$.)

A straightforward, but not particularly elegant, solution to this problem is to build some structure into the data sequence (e.g., by inserting parity-check bits) so as to facilitate the elimination of false data sequences. This approach requires no major changes in the algorithm itself. Another approach is to try to modify the algorithm so that both the encoder and the decoder update their states using a next-state function with finite look-ahead and finite look-back

$$\sigma_i = f(x_{i-k}, \cdots, x_{i+k'}), \tag{13}$$

where k, k' are fixed integers. Clearly, an algorithm with such a next-state function is noncatastrophic, with a maximum synchronization delay of $k + k' + 1$ symbols. The algorithm here has a next-state function of the form $\sigma_i = f(\sigma_{i-1}, x_i)$, which has no look-ahead, but possibly infinite look-back. We have found no clean way of modifying the algorithm so as to eliminate the potentially infinite memory from its next-state function.

We should note that there are algorithms (e.g., [9], [10]), for which the next-state function is in the form (13), with one important difference: for these algorithms, each argument of the next-state function (the terms $x_{i-k}, \cdots, x_{i+k'}$) is a block of q symbols (for some fixed integer q) from the channel alphabet. Unfortunately such a next-state function does not provide protection against misframing (i.e., incorrectly deciding the block boundaries); hence, these codes, too, are subject to catastrophic error propagation.

V. Conclusion

Nonfixed-rate codes require buffering if one wishes to use the channel synchronously, and one has to worry about the consequent buffer overflow problems. For this reason there is a bias towards fixed-rate codes in practice. Such a bias is hard to justify in systems where the source sequence is highly redundant and one uses a variable-length source code (such as a Huffman code) to remove part of that redundancy, since buffering is required then whether the channel coding is fixed-rate or not.

Fixed-rate codes do not present a buffer-overflow problem, but they may require considerably more implementation complexity. Consider, for example, the fixed-rate codes proposed in [7], [8], and the ones pioneered by Adler, Hassner, and Coppersmith [9], [10], [11]. These codes are constrained to work at rational rates; and, to construct a code with rate p/q, one needs to work with a q-step state-diagram for the given channel. Since the q-step diagram contains on the order of 2^{qC} states for a channel with capacity C, the implementation complexity of these codes is exponential in q. The constructions in [9], [10], [11] have an even higher complexity because they apply state-splitting (an elegant procedure invented by Marcus [12]) to the q-step diagram, resulting in a further increase in the final number of states. The fact that the complexity is exponential in q places a practical limit on how close one can get to the channel capacity with rational rates of the form p/q.

In contrast, the algorithm here never has to deal with extended channels no matter how close the desired coding rate is to channel capacity. In order to achieve higher rates, one need only run the same algorithm using a higher precision.

References

[1] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Urbana, IL: Univ. of Illinois Press, 1963.

[2] B. McMillan, "The basic theorems on information theory," *Ann. Math. Stat.*, vol. 24, pp. 196–219, June 1953.

[3] F. R. Gantmacher, *The Theory of Matrices, vol. II*. New York: Chelsea, 1959.

[4] G. N. N. Martin, G. G. Langdon, Jr., S. J. P. Todd, "Arithmetic codes for constrained channels," *IBM J. Res. Develop.*, vol. 27, pp. 94–106, Mar. 1983.

[5] S. J. P. Todd, G. G. Langdon, Jr., G. N. N. Martin, "A general fixed rate arithmetic coding method for constrained channels," *IBM J. Res. Develop.*, vol. 27, pp. 107–115, Mar. 1983.

[6] H. Sato, "On the generation of run-length constrained codes by arithmetic coding," *IEICE (Inst. Electron., Inform. Comm. Engineers), Technical Report*, vol. 87, IT 87-94, pp. 25–30, 1987 (in Japanese).

[7] P. A. Franaszek, "Construction of bounded delay codes for discrete noiseless channels," *IBM J. Res. Develop.*, pp. 506–514, July 1982.

[8] A. Lempel and M. Cohn, "Look-ahead coding for input-restricted channels," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 933–937, Nov. 1982.

[9] R. L. Adler, D. Coppersmith, and M. Hassner, "Algorithms for sliding block codes," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 5–22, Jan. 1983.

[10] B. Marcus, "Sofic systems and encoding data," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 366–377, May 1985.

[11] R. Karabed and B. H. Marcus, "Sliding-block coding for input-restricted channels," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 2–26, Jan. 1988.

[12] B. Marcus, "Factors and extensions of full shifts," *Monats. Math.*, vol. 88, pp. 239–247, 1979.

[13] A. Norris and D. S. Bloomberg, "Channel capacity of charge-constrained run-length limited codes," *IEEE Trans. Mag.*, vol. Mag-17, pp. 3452–3455, Nov. 1981.

[14] R. E. Blahut, *Principles of Information Theory*. Reading, MA: Addison-Wesley, 1987.

[15] C. B. Jones, "An efficient coding system for long source sequences," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 280–291, May 1981.

[16] K. J. Kerpez, "Run length codes from source codes via maximum entropy probabilities," *Abstracts of Papers, IEEE Int. Symp. Inform. Theory*, Kobe, Japan, 1988.

[17] E. Çinlar, *Introduction to Stochastic Processes*. Englewood Cliffs, N.J.: Prentice-Hall, 1975.