Coding for constrained channels: A comparison of two approaches

by Peter A. Franaszek

We discuss the relation between some early techniques for constrained channel coding and more recent ones adapted from the mathematical area of symbolic dynamics. A primary difference between the two is that the latter focus on issues of code existence, whereas the former were primarily concerned with code construction and optimality.

1. Introduction

One of the more notable developments in information theory within the last few years has been the discovery [1] of the relationship between the recent work in the mathematical area of symbolic dynamics and what is often termed channel coding. Aside from the work [1] by Adler, Coppersmith, and Hassner (referred to here as ACH), this has led to such significant results as those of Marcus [2], and in general this development has substantially improved the mathematical underpinnings of this coding area. Constrained channel codes are used in a wide variety of applications. These include digital transmission on electrical or fiber-optic media, as well as digital recording. Each such system involves different tradeoffs which affect the form of suitable signal sequences and hence the design or choice of the channel codes. A consequence is that new codes continue to be developed, and construction techniques remain of practical interest.

In their paper, ACH give a constructive proof for the existence of minimum block length sliding block codes for channels with finite memory and provide a related method for generating codes. They note the existence of other noiseless coding methods, including what we term bounded delay codes [3–5], and mention that it would be worthwhile to clarify the relation between these two approaches. This is the topic of this paper. The two techniques yield codes of somewhat different form (ACH employ no lookahead). However, we will show that for any code that can be constructed via the ACH technique, there is a corresponding one obtainable via the bounded delay (referred to here as BD) method. The BD technique provides algorithms for constructing codes which for example minimize a parameter related to implementational complexity and error propagation.

We proceed in our comparison of the two code generation methods by considering each step in the procedures (with references to the original papers for

many details), and illustrate the discussion, as do ACH, via the example of (2, 7) run length coding at rate 1/2. The terms used here will generally be standard ones from coding practice, although the ACH terminology is used occasionally. Except for a brief discussion of construction differences for the infinite memory case, we generally assume, as do ACH, that the channels are of finite memory.

2. State transition matrices and approximating eigenvectors

ACH begin, as does the BD technique, with a finite state machine description of the channel constraints. **Figure 1** illustrates the state transition diagram for the (2, 7) constraints. ACH pick a coding rate $C' \leq C$, the channel capacity, and a block or word size W. Next is a search for an approximating eigenvector for a channel matrix $\mathbf{T}$ of a channel where each transition corresponds to W transitions in the original representation of the constraints. These steps follow what has been more or less common coding practice (although early results such as [6–8] were restricted to special classes of eigenvectors). ACH note that the integer programming algorithm they use for obtaining an eigenvector was described earlier in [4], and in fact this is a generalized version of one obtained in [6]. This algorithm provides a method for obtaining an approximating eigenvector with an eigenvalue corresponding to a lower bound for a parameter related to complexity, as is discussed below. The algorithm finds (if one exists) a vector corresponding to a chosen value of a lookahead parameter M (defined in Section 4). The approach is to apply the algorithm for $M = 1, 2, \cdots$. Failure to obtain a vector for a suitably low value of M, usually a small integer, means that no practical code exists which corresponds to the chosen parameters. ACH note that the existence of an eigenvector for some value of M is a consequence of the classical Perron–Frobenius theory [9].

In their illustration of the procedure via the (2, 7) coding example, ACH pick a rate of 1/2 and a block or word length of 2. That is, the encoding is for one bit at a time which will be mapped onto code words each comprising two state transitions in the original constraints. Available code words correspond to the following transitions between channel states σ_i:

$\sigma_0 \rightarrow \sigma_2$,
$\sigma_1 \rightarrow \sigma_0\sigma_3$,
$\sigma_2 \rightarrow \sigma_0\sigma_1\sigma_4$,
$\sigma_3 \rightarrow \sigma_0\sigma_1\sigma_5$,
$\sigma_4 \rightarrow \sigma_0\sigma_1\sigma_6$,
$\sigma_5 \rightarrow \sigma_0\sigma_1\sigma_7$,
$\sigma_6 \rightarrow \sigma_0\sigma_1$,
$\sigma_7 \rightarrow \sigma_1$.

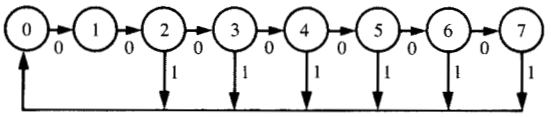

Figure 1

Illustration of (2, 7) run-length-limited constraints.

Table 1 Encoding state transition table (from [1], adapted with permission; © 1983 IEEE).

						$x_n = 0$		$x_n = 1$
					0^1	$\rightarrow$	2^2	2^4
					0^2	$\rightarrow$	2^1	2^3
		5^1	4^1	3^1	2^1	1^1	$\rightarrow$	0^2	0^1
						1^2	$\rightarrow$	3^1	3^3
						1^3	$\rightarrow$	3^2	3^4
7^1	6^1	5^2	4^2	3^2		2^2	$\rightarrow$	1^2	1^3
						2^3	$\rightarrow$	4^1	1^1
						2^4	$\rightarrow$	4^2	4^3
						3^3	$\rightarrow$	5^1	1^1
						3^4	$\rightarrow$	5^2	5^3
						4^3	$\rightarrow$	6^1	1^1
						5^3	$\rightarrow$	7^1	1^1

For example, state σ_5 admits three possible code words (01, 10, 00) that take the channel to σ_0, σ_1, and σ_7, respectively.

The approximating eigenvector used by ACH in their example is given by $V = \{v_i\} = (2, 3, 4, 4, 3, 3, 1, 1)$, where v_i is termed the *weight* of σ_i. We note that $v_i \leq 2^2$, so that in our terminology the lookahead or *delay-bound* for M is no less than 3, a fact we use below. Loosely speaking, the weight v_i corresponds to the amount of information that can be transmitted from σ_i.

3. State splits or tree partitions

The next step in the ACH procedure is state splitting (of a form introduced in [10]), a technique which splits each σ_i into no more than v_i substates ρ_i^j so that each ρ_i^j has successors of appropriate total weight (in this case 2). Unnecessary transitions (those not required to obtain sufficient weight) are eliminated. Transitions between the ρ_i^j are then assigned information symbols. This yields Table VII in ACH, used to obtain **Table 1**, where ρ_i^j are represented simply by i^j. Note that a necessary (but not sufficient) condition for ρ_i^j to have a transition to ρ_k^m is that σ_i have one to σ_k. Decoding is done by determining which ρ_k^m are traversed.

ACH observe that there are a variety of possible splitting outcomes, depending on choices taken in their algorithm, but in practice transitions between the states σ_i permit the determination of the path taken between the substates ρ_i^j in encoding (in the ACH terminology the mapping is said to have resolving blocks), so that it is possible to decode. They also show that if appropriate restrictions are imposed in the splitting and construction procedure, resolving blocks are guaranteed to exist. ACH thus obtain the important new result that if the channel is of finite memory, then for any $(\alpha/N) \leq C$ there exist rate (α/N) sliding block codes with block size N.

The construction procedure described by ACH involves a sequence of steps at each of which there may be several choices. Thus, design of a code typically entails the examination of a (possibly large) number of splits (as well as block sizes, as the minimum block size is not necessarily optimal), in an attempt to minimize code complexity and error propagation. The procedure is based on conditions ACH show are sufficient (for obtaining resolving blocks), but which are not shown to be necessary. It is not immediately obvious (and ACH do not mention) whether all splits (and in particular the optimal ones) for which resolving blocks exist can be obtained by appropriate choices in their algorithm. Another issue of some interest is when to stop the search for the best available split. These are issues which the BD method attempts to address.

The BD method is based on constructs termed *independent paths* (*IP*s), introduced in [3], which we define here in a manner which serves to link the two methods. For sake of definition, we first assume that some arbitrary state splitting procedure has been carried out to the point where all substates ρ_i^j are of weight one.

Definition An $IP(i, j, D)$ is the successor tree to depth D from a substate ρ_i^j. If $D = (M - 1)$, this is simply referred to as $IP(i, j)$.

Let $L_i(D)$ denote the leaves $\{\sigma_i\}$ of the state successor tree $T(i, D)$. Both $T(i, D)$ and $IP(i, j, D)$ are constructed so that each leaf represents a distinct sequence of D transmitted symbols starting from σ_i. Thus, $L_i(D)$ may contain several replications of some states, but these are considered distinct. Note that if $IP(i, j, D)$, $IP(i, k, D)$ share no members of $L_i(D)$, then any encoded sequence of length D can be used to differentiate between the two. Conversely, if these two *IP*s share a member of $L_i(D)$, then there exists a sequence of length D which does not permit such differentiation. *IP*s rooted at a state σ_i are said to be distinguishable at depth D if they share no members of $L_i(D)$.

In the BD method, the encoder is viewed as following an *IP* for one word (hence the term path), then switching to an *IP* from the destination state. Another way of regarding this is that the encoder touches a sequence of *IP* tree roots (or equivalently ρ_i^j). Decoding is done by determining which *IP*s were traversed. The channel's finite memory ensures that a sliding block decoder can recover the sequence of traversed channel states. It follows that a necessary and sufficient condition for being able to recover *IP* sequences with such a decoder is that all *IP*s from each state be distinguishable at some finite D. Note that the sequence of traversed *IP*s can be determined if and only if it is possible to recover the sequence of *IP* roots or substates ρ_i^j. That is, this is equivalent to the mapping having resolving blocks in the ACH terminology.

The following is a brief summary of the *IP* construction procedure [3–5]. After choosing a code rate, block size, and approximating eigenvector with components ν_i, successor trees $T(i, j, D)$ are formed from each σ_i. Members of this tree set are interrelated by stationarity conditions or constraints which impose orderings on successors of the ρ_i^j. Each state σ_i is viewed as being comprised of ν_i subunits ρ_i^j. A partitioning of the leaf sets $L_i(D)$ into disjoint subsets of appropriate weight assigns to each $\rho_i^j \in \sigma_i$ successors of suitable weight from states σ_k, which are successors of σ_i. As noted above, the existence of such a partition is a necessary and sufficient condition for recovering the sequence of traversed ρ_i^j with a sliding-block decoder which has available to it the $(D + 1)$ most recently occupied states.

The parameter D is related to both complexity and error propagation, so that it is usually desirable to obtain a partition for minimum D. If none exists for a suitably small value (usually a small integer) of this parameter, this leads to a resort to such measures as changing the block size, the channel constraints, or the code rate. The result obtained by ACH guarantees the existence of a partition for finite D, in particular for D no larger than $\Sigma\nu_i$. A lower bound for D follows from that on M and is related to Max ν_i, as discussed below.

In other words, ACH employ a systematic procedure for state splitting which if appropriately constrained terminates successfully, yielding a finite value for D. In contrast, the BD technique attempts partitions for fixed values of D. Each such attempt is a straightforward but tedious problem (the unconstrained partitioning of a set of leaves is *NP*-complete), and inability to obtain a satisfactory value for D means that no practical code of this type exists for these parameters. This divide and conquer approach is essentially patterned after that used to obtain the variable-length codes in [7].

4. Code formats and message assignments

Code mappings of various forms are described in [4, 5]. Of these, the one we discuss here is obtained by assigning message characters to the first transitions of *IP*s starting

at leaf state of successor trees of depth $(M-1)$. The method includes an algorithm, based on a lexicographical ordering of message vectors, which is shown to yield assignments such that, using lookahead of M characters, the appropriate sequence of *IP*s is taken when a given character string is to be encoded. The assignment can be varied by changing the successor ordering within *IP*s. More formally, the encoding is given by

$$[\underline{V}(0), \underline{B}(0)] \rightarrow \sigma_{i(1)}, \tag{1}$$

where $\underline{V}(0)$ is the sequence of M states $[\sigma_{i(t-M+1)}, \sigma_{i(t-M+2)}, \cdots, \sigma_{i(0)}]$ most recently occupied by the channel, and $\underline{B}(0)$ are the M information characters to be encoded next. M is chosen so that $(M-1)$ is not less than D. This ensures that $\underline{V}(0)$ is sufficient to determine which *IP* was followed from $\sigma_{i(t-M+1)}$. Each $\rho^m_{i(0)}$ is associated with a unique sequence of $(M-1)$ characters to be transmitted. Knowledge of the next M characters is then sufficient to determine the transition $\rho^m_{i(0)} \rightarrow \rho^n_{j(1)}$, and thus the next code word. One property of the above form is that it is unnecessary to keep track of substates ρ^j_i. Note also that $(D+1)$ state transitions are required for decoding.

The above discussion framed the problem in terms of the parameter D, then introduced the lookahead parameter M. This facilitates comparison with the approach of ACH. The constructions in [3–5] are described in terms of fixed but arbitrary M, with $D \leq (M-1)$ required in order for *IP*s to be distinguishable. A lower bound for M is obtained, $(M^* - 1) \geq r$, where r is the least integer such that $\alpha^r \geq \nu_i$, $\forall i$, and which still permits an appropriate eigenvector. The parameter α denotes the number of bits per word. Another way to obtain the lower bound is to note that if *IP*s are distinguishable, there exists one or more which has a leaf at depth D of weight corresponding to r. Otherwise a lower value of r could have been chosen in the algorithm for generating the eigenvector.

Each code of the ACH form corresponds to one of bounded delay type which can be obtained as follows: (1) Use the state split to form *IP*s. (2) Consider the encoded character corresponding to each word transition $\rho^j_i \rightarrow \rho^m_r$. Assign this character to all transitions from the substate leaves of $IP(i, j)$ which are descendants of ρ^j_i via the transition to ρ^m_r.

Going from the root of $IP(i, j)$ to a leaf requires $(M-1)$ transitions. Applying the second step in the above procedure $(M-1)$ times results in the assignment of a character sequence of length $(M-1)$ to each leaf $\rho^m_{i(0)}$ in the mapping. Note that in each case the same information can be viewed as being carried by the same transition $\rho^j_i \rightarrow \rho^m_r$. In the bounded delay form, the transition is taken because of knowledge from lookahead that the same information character is to be encoded "later." It can be shown that any bounded delay code corresponds to one with no lookahead which can be obtained by reversing the above transformation.

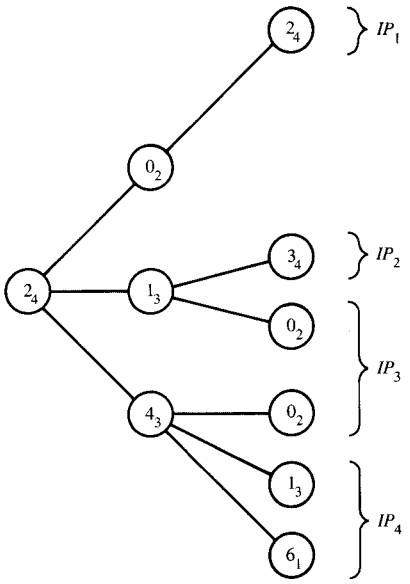

Figure 2

*IP*s from state σ_2. Here l_j refers to σ_j of weight j.

We now illustrate some of the above notions via the (2, 7) coding example. The chosen eigenvector is (2, 3, 4, 4, 3, 3, 1), $\nu_i \leq 2^2$, so that $D \geq 2$. **Figure 2** shows $T(2, 2)$, the state successor tree of depth 2 from σ_2. The leaves $L_2(2)$ partition into sets of weight no less than 4 (that required for *IP*s with $M = 3$). The *IP*s in Figure 2 correspond to the split of σ_2 in the ACH coding example. There are four *IP*s rooted at σ_2. Note that two transitions are sufficient to determine which one is being followed. For example, the sequence $\sigma_2 \rightarrow \sigma_1 \rightarrow \sigma_0$ means that IP_3 is being followed. Other states, however, require $D = 3$, so that $M = 4$. **Figure 3** shows the substate successor tree from ρ^2_2, which for depth 2 corresponds to IP_3 in Figure 2. This was obtained from Table 1. Each transition in Figure 3 is marked by an information symbol. Thus, for example, $\rho^2_2 \rightarrow \rho^2_1$ (represented as $2^2 \rightarrow 1^2$ in the figure) corresponds to the encoded character 0.

If the bounded delay form is to be employed, occupation of ρ^2_1 after, say, the transition sequence $\rho^2_2 \rightarrow \rho^3_1 \rightarrow \rho^2_3 \rightarrow \rho^2_1$ indicates that the information sequence (100) "is to be encoded." Note that the transitions $\sigma_2 \rightarrow \sigma_1 \rightarrow \sigma_3 \rightarrow \sigma_1$ are sufficient to show that

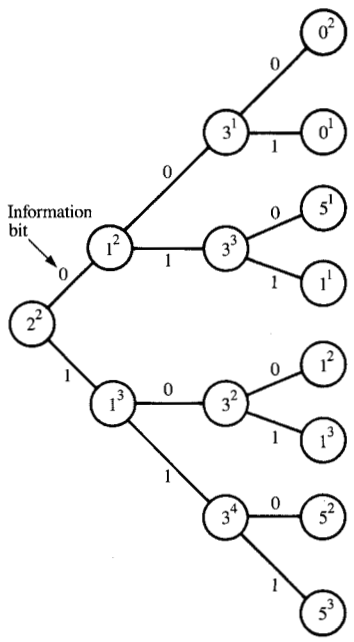

Figure 3

Successor tree from ρ_2^2 in the ACH example.

ρ_2^2 was occupied in σ_2 (i.e., that IP_3 is being followed). The fact that the sequence (100) is to be encoded then determines the identity of the occupied substate in σ_1, namely ρ_1^2. An additional information symbol, for a lookahead of $M = 4$, then determines the next transition to be taken. A sequence of transmitted information symbols leads to a traversal of the same substates as in the ACH example.

The above discussion is posed in terms of state transitions. However, a sliding-block decoder operates by examining a sequence of, say, R code words (each corresponding to a state transition), and emitting a symbol β representing α bits. This is done once for each received code word:

$$[w_{i-R+1}, w_{i-R+2}, \cdots, w_i] \rightarrow \beta_i . \quad (2)$$

A given sequence $[w_{i-R+1}, \cdots, w_i]$, however it arises (in terms of state transitions), must map onto the same value β, a property termed state-independent decodability [2, 6]. The parameter R is an upper bound for the number of information symbols that are decoded erroneously as the result of a single channel error. For the class of codes considered here, R is also sufficient for *IP*s to be distinguishable, so that $R \geq (D + 1)$.

If the channel is of finite memory L, then L code words $w_i, w_{i+1}, \cdots, w_{i+L-1}$ are sufficient to identify the state entered at $(i + L - 1)$. A decodable sliding-block code may be obtained by forming *IP*s of depth D, then using a decoder which examines word sequences of length $R = (L + D + 1)$. Since *IP*s of depth $(L + D)$ are also distinguishable, the code may be viewed as a mapping of the form given by Equation (1), with $M = (L + D + 1)$, in which assignments are state-independently decodable.

More generally, for channels of either finite or infinite memory, *IP*s of depth $(M - 1) \geq D$ are formed, with M a parameter, and an attempt is made to obtain a mapping for which decoding is state-independent. Increasing M increases the freedom in forming *IP*s. It also tends to decrease the number of like word sequences (aliases) produced by different state transitions. If the channel is of finite memory, then $M \leq (L + D + 1)$ is sufficient, and usually $M < (L + D + 1)$. For channels with infinite memory, increasing M may not necessarily yield a state-independently decodable code.

5. Discussion and conclusion

A question of some interest is the degree of improvement provided by the BD or ACH methods over older techniques such as those described in [6–8]. The latter methods are similar in approach (i.e., choice of a fixed rate, an integer programming algorithm for obtaining an approximating eigenvector), but were eventually shown [3] to be limited to restricted classes of eigenvectors. For these codes, a subset of the channel states is designated as a principal set, from which words start and at which they end. Each state in the principal set is required to have sufficient words (of varying length) to satisfy the Kraft inequality. The codes are optimized (by minimizing the maximum word length) via a dynamic programming version of the algorithm described in Section 2. The variable-length method often yields striking improvements over block codes, as can be seen from the tables included in [8]. For example, consider the $(2, K)$, $K \geq 7$, constraints at rate 1/2. A variable-length (2, 7) code [11] is obtained with a totality of six channel words. In contrast, a fixed-length code for the easier (2, 8) constraints requires 11 information bits per word and thus at least 2^{11} words. Advantages of the BD or ACH methods over the variable-length approach are less clear, because the latter yields optimal or close to optimal codes for many examples. ACH note that the (2, 7) code they derive is somewhat more complex than that used, for example, in the IBM 3380 disk drive [11–13], obtained via the variable-length method. Horiguchi and Morita [14] improved the variable-length technique by modifying the decoding rule of [8]. As a result, they obtained improved codes for some parameters, including

a (1, 7) rate 2/3 code comparable (error propagation is one bit greater) to that obtained via the BD or ACH methods [15, 16]. It is not clear whether the newer approaches yield major improvement for cases of practical interest. However, as shown in [3], there exist channels for which the variable-length codes cannot reach full capacity with a finite maximal word length. See [17] for a recent discussion of some properties of such codes and their corresponding constrained sequences.

Most of the above-mentioned codes are for channels with finite memory. Also common in practice are constraints which yield channels of infinite memory, typically the imposition of spectral nulls, most often at zero frequency. See [6, 18, 19] for examples of such codes which have been applied respectively in electrical and fiber-optic data transmission systems, and digital magnetic recording on tapes. Reference [6] introduced the notion of forcing a null at zero frequency by bounding the running digital sum (RDS), and includes results on code assignments to obtain state-independent decoding in state-dependent block codes. Reference [19] derives the class of zero-modulation (ZM) codes, as well as various properties of constrained codes and sequences, including observations based (as are some of ACH's), on the Perron–Frobenius theory.

This paper discusses the relation between constrained coding techniques adapted by ACH from the mathematical area of symbolic dynamics and some previous work in coding theory. The former include their recent constructive proofs [10] for the existence of homeomorphisms of a type suitable for coding between equientropic sequences. These were adapted by ACH to prove the existence of minimum block length sliding block codes for finite-memory constrained channels, and to obtain a construction technique based on state-splitting. The resulting codes are of somewhat different form from those available via the earlier bounded delay (BD) methods, but each such code corresponds to one that can be obtained by the BD approach. The latter method includes procedures which yield codes corresponding to a minimum value for a parameter related to complexity and error propagation.

Applications continue to arise for which the design of suitable codes still requires much computation and examination of alternatives. This suggests that algorithms for constrained code design will continue to be a fruitful area of investigation.

Acknowledgment

The author is indebted to J. L. Wolf for discussions on symbolic dynamics as well as for many helpful comments regarding this manuscript.

References

1. R. L. Adler, D. Coppersmith, and M. Hassner, "Algorithms for Sliding Block Codes," *IEEE Trans. Info. Theory* **IT-29,** 5–22 (1983).
2. B. Marcus, "Sofic Systems and Encoding Data," *IEEE Trans. Info. Theory* **IT-31,** 366–377 (1985).
3. P. A. Franaszek, "On Future-Dependent Block Coding for Input-Restricted Channels," *IBM J. Res. Develop.* **23,** 75–81 (1979).
4. P. A. Franaszek, "A General Method for Channel Coding," *IBM J. Res. Develop.* **24,** 638–641 (1980).
5. P. A. Franaszek, "Construction of Bounded Delay Codes for Discrete Noiseless Channels," *IBM J. Res. Develop.* **26,** 506–514 (1982).
6. P. A. Franaszek, "Sequence-State Coding for Digital Transmission," *Bell Syst. Tech. J.* **47,** 113–157 (1968).
7. P. A. Franaszek, "On Synchronous Variable Length Coding for Discrete Noiseless Channels," *Info. Control* **1-J,** 155–164 (1969).
8. P. A. Franaszek, "Sequence-State Methods for Run-Length Limited Coding," *IBM J. Res. Develop.* **14,** 376–383 (1970).
9. F. R. Gantmacher, *The Theory of Matrices*, Vol. II, Chelsea Publishing Co., New York, 1959.
10. B. Marcus, "Factors and Extensions of Full Shifts," *Monatshefte für Math.* **88,** 239–247 (1979).
11. P. A. Franaszek, "Run Length Limited Variable Length Coding with Error Propagation Limitation," U.S. Patent 3,689,899, 1972.
12. J. S. Eggenberger and P. Hodges, "Sequential Encoding and Decoding of Variable Word Length, Fixed Rate Data Codes," U.S. Patent 4,115,768, 1978.
13. J. M. Harker, D. W. Brede, R. E. Pattison, G. R. Santana, and L. G. Taft, "A Quarter-Century of Disk File Innovation," *IBM J. Res. Develop.* **25,** 677–689 (1981).
14. T. Horiguchi and K. Morita, "An Optimization of Modulation Codes in Digital Recording," *IEEE Trans. Magnetics* **MAG-12,** 740–742 (1976).
15. P. A. Franaszek, "Efficient Code for Digital Magnetic Recording," *IBM Tech. Disclosure Bull.* **23,** 4375 (February 1981); U.S. Patent 4,488,142, 1984.
16. R. Adler, M. Hassner, and J. Moussouris, "Method and Apparatus for Generating a Noiseless Sliding Block Code for a (1, 7) Channel with Rate 2/3," U.S. Patent 4,413,251, 1983.
17. T. D. Howell, "Statistical Properties of Selected Recording Codes," *IBM J. Res. Develop.* **33,** 60–74 (1989).
18. A. X. Widmer and P. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code," *IBM J. Res. Develop.* **27,** 440–451 (1983).
19. A. M. Patel, "Zero-Modulation Encoding in Magnetic Recording," *IBM J. Res. Develop.* **19,** 366–378 (1975).

Received February 15, 1989; accepted for publication August 31, 1989

Peter A. Franaszek *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Dr. Franaszek is manager of Systems Theory and Analysis in the Computer Sciences Department at the Thomas J. Watson Research Center. He received an Sc.B. degree from Brown University in 1962, and M.A. and Ph.D. degrees from Princeton University in 1964 and 1965, respectively. Dr. Franaszek's interests include analytical and design issues in computer system organization, algorithms, and communication networks and coding. He has received IBM Outstanding Innovation Awards for his work in the areas of

algorithms, interconnection networks, and constrained coding. Dr. Franaszek was also the recipient of two IBM Corporate Awards for his work in the latter area. He was named the recipient of the 1989 Emanuel R. Piori Award of the Institute of Electrical and Electronics Engineers for his contribution to the theory and practice of digital recording codes. During the academic year 1973–1974, he was on sabbatical leave from the Thomas J. Watson Research Center to Stanford University as a Consulting Associate Professor of Computer Science and Electrical Engineering. Prior to joining IBM in 1968, he was a member of the technical staff at Bell Telephone Laboratories. Dr. Franaszek is a member of the Institute of Electrical and Electronics Engineers, Tau Beta Pi, and Sigma Xi.