Enumerative modulation coding with arbitrary constraints and postmodulation error correction coding for data storage systems

Masud Mansuripur

Masud Mansuripur, "Enumerative modulation coding with arbitrary constraints and postmodulation error correction coding for data storage systems," Proc. SPIE 1499, Optical Data Storage '91, (1 July 1991); doi: 10.1117/12.45930

SPIE.

Event: Optical Data Storage, 1991, Colorado Springs, CO, United States

Enumerative Modulation Coding with Arbitrary Constraints and Post-Modulation Error Correction Coding for Data Storage Systems

M. Mansuripur

Optical Sciences Center, University of Arizona, Tucson, Arizona 85721

ABSTRACT

Modulation constraints of practically any degree of complexity can be described by a state transition table with a finite number Ω of states. Examples include all $(d,k;c)$ codes (where $\Omega \leq 2(k+1)(2c+1)$), more general codes with run-length limitations, and run-length limited codes which exclude certain bit-patterns. From the state transition table we construct a trellis diagram for code words of arbitrary length L_0. If desired, the trellis may be confined in the beginning and/or at the end to a subset of states. We then show a simple method of enumeration that assigns a number to each code word in the trellis according to its lexicographic order. All the necessary information for enumerative encoding and decoding of binary data will be subsequently stored in an array of size $L_0 \times \Omega$; both encoding and decoding can be achieved with a few simple operations using this table. In short, arbitrarily long blocks of data can be encoded into sequences that satisfy arbitrary constraints, with algorithms that are easy to implement. Since no additional constraints are imposed, the rates approach Shannon's noiseless channel capacity in the limit of long sequences. We present ideas for correction of random errors that occur in modulated sequences, so that errors in readout can be corrected prior to demodulation. These post-modulation error correction codes are necessary when modulation code words are long, in which case small errors can destroy large quantities of data. Also introduced in this paper is a simple, efficient algorithm for burst-error-correction. The primary application of the ideas of this paper is in the area of data encoding/decoding as applied in magnetic and optical data storage systems

1. INTRODUCTION

The use of modulation codes in magnetic and optical data storage systems is widespread and the advantages of modulation coding are well known[1-7]. Usually a 1 is used to represent a transition between the up and down states of the recording waveform, while a 0 represents no change in the waveform. An unconstrained sequence of 1's and 0's, however, is not acceptable, nor is it desirable, in practice. For instance, a long sequence of uninterrupted 0's results in loss of synchronization when the data is self-clocking. Or the imbalance between the up and down states of the recording waveform may result in significant charge accumulation in the electronic circuitry, thus forcing the system beyond acceptable levels of operation. The objective of modulation coding is to create a one-to-one correspondence between sequences of user data (which are usually streams of random binary digits) and constrained binary sequences. The nature of the constraints imposed on the modulation signal must be determined by the system designer and is dependent on the particular characteristics of the system under consideration. A typical set of restrictions imposed on the recording sequence is the so-called $(d,k;c)$ constraint. Here d is the minimum allowed run-length of zeros, k is the maximum allowed run-length of zeros, and c is the maximum charge that the recording waveform is permitted to accumulate. In addition, certain sequences of bits may be reserved for special purposes (such as synchronization or signaling the beginning of a block) and thus banned from appearing in the modulated waveform.

The minimum run-length constraint d is somewhat more subtle than the other constraints and needs further elaboration. The physical separation between successive transitions on the recording medium cannot be made arbitrarily small. The minimum distance between transitions is a function of the structural and/or magnetic properties of the medium, as well as the characteristics of the read/write head. The gap-width of the magnetic recording head and the wavelength of light in optical recording, together with the structural/magnetic features of the recording medium, determine the minimum distance Δ between successive transitions. It is useful to measure distance along the track in units of Δ, so in the following discussion, density should be understood as the number of bits per Δ. Now, if there are no constraints on the minimum run length of zeros (i.e., if $d = 0$),

0-8194-0608-2/91/$4.00

each modulation bit will occupy one interval of length Δ. Since the number of modulation bits is always greater than the number of data bits, the overall recording density will become less than one data bit per Δ. On the other hand, if $d \neq 0$, one can pack d+1 modulation bits in every interval of length Δ. When the code parameters are chosen properly, the higher density of modulation bits could translate into higher density of recording for data bits, so that the overall density will become greater than one data bit per Δ. The price of this increase in capacity is, of course, the reduced period of time available to each modulation bit. If τ is defined as the time that the read/write head dwells on an interval of length Δ, then the time window available to each modulation bit will be $\tau/(d+1)$.

The purpose of the present paper is to introduce a general algorithm for modulation coding with arbitrary constraints. This algorithm which uses enumeration to encode blocks of data into constrained sequences, is much more powerful than an earlier scheme (also based on enumeration) proposed by Tang and Bahl[8]. In contrast to the latter scheme which applies only to (d,k) constraints, the new algorithm can handle any constraint or set of constraints that can be expressed in terms of a state transition table. In addition, our algorithm solves the cascading problem which complicated all previous schemes and required the addition of "merging" bits between successive blocks.

In Section 2, we will give several examples of (d,k) and $(d,k;c)$ constraints and their corresponding state transition tables. We also give an example of a (d,k) code which excludes a specific pattern of bits for synchronization purposes. Although the examples in Section 2 are for certain representative values of parameters, the techniques used for constructing the state transition tables are quite general, and the reader should be able to apply the methods to any set of parameters.

After the state transition table corresponding to a given set of constraints has been constructed, one must choose a parameter L_0 for the block length of the modulation code words. Any choice of L_0 can be handled by the algorithm, but longer code words are preferred since they give a capacity which is arbitrarily close to Shannon's noiseless-channel capacity[9] for the given set of constraints. The encoding and decoding routines are based on enumeration[10,11], and map blocks of user data of fixed length N_0 to modulation code words of fixed length L_0. Both encoding and decoding routines utilize an array A of pre-calculated numbers. The array A has dimensions $L_0 \times \Omega$ where Ω is the total number of states in the state transition table. The mappings are of fixed-to-fixed length block type, and the beginning state is always the same as the final state, so that errors cannot propagate beyond the block boundaries. The trellis diagram, discussed in Section 3, helps in understanding this point as well as in constructing the array A. The encoding and decoding algorithms are then described in Section 4.

In conjunction with the proposed modulation technique, two error correction schemes will also be described. The first type has to do with burst-error-correction and is discussed in Section 5. Here one takes advantage of the fact that the position of a block of data affected by a long burst of errors is known to the decoder, simply because modulation constraints within that block are seriously violated. A simple construction based on parity check bits and capable of restoring the lost block will be introduced in Section 5. The second type of error correction deals with random errors that occur rather infrequently and corrupt modulation code words in a few, a priori unknown, locations. Some of these errors may be readily detectable (and perhaps even correctable), because they violate the modulation rules. Depending on the severity of the constraints, however, this kind of error correction may or may not have the desired power in practice. In any event, a Viterbi decoder[12-14] can be used to find the closest code word (in the Hamming sense) to the readout sequence. In this context, the applicability of Viterbi's algorithm for minimum distance decoding arises from the fact that modulation coding is based on state transition tables. This point will be further elaborated in Section 6. The other, and perhaps more powerful, alternative for random error correction within a modulated block is the use of an appropriate block error-correcting-code, such as a Reed-Solomon code[15,16]. Here, depending on the probable number of errors within a block of length L_0, the ECC scheme generates a certain number of check bits, say N_1. These check bits do not necessarily satisfy the modulation constraints and, therefore, must themselves be modulated. Let us denote by L_1 the length of the block of N_1 check bits after being modulated. Since encoded blocks always begin and end in the same state, there shall arise no problems during recording and readout if the block of length L_1 immediately follows the block of length L_0. In certain situations it might be desirable to apply another level of error correction to the block of length L_1, and to record the new set of check bits after modulating them. Indeed the process may be repeated any number of times. After the modulated block of data, followed by one or more blocks of modulated check bits, has been formed, one might

add a special block of length L_s to signal the end of the stream. All in all, the sequence of user data of length N_0 has been mapped onto a sequence that satisfies the modulation constraints and has a total length $L = L_0 + L_1 + L_2 + \cdots + L_s$. In readout, this block of length L is separated into its sub-blocks, its various levels of error correction are demodulated, and the corresponding errors, if any, are corrected. Finally, the block of length L_0 is demodulated and the original N_0 bits of user data are recovered. The details of this procedure are described in Section 7.

Section 8 gives several examples of codes with various constraints and discusses their performance. The final section contains some closing remarks, including a suggestion for experimentally determining the proper set of modulation constraints for a given data storage device.

2. THE STATE-TRANSITION TABLE

This section demonstrates, by way of examples, the construction of state-transition tables from the rules and constraints that a modulation code must satisfy.

Example 1 : *A* (d,k) code is one in which the run-length of zeros has a minimum equal to d and a maximum equal to k. Figure 1 shows the state transition diagram for a (1,3) code. (Extension to arbitrary values of d and k is straightforward.) *A* circle in this diagram represents a state and the identity of that state is written inside the circle. The states in Figure 1 are S_1, S_2, S_3, and S_4. Each state leads to at least one and at most two states. If the transition from one state to another is indicated by a solid line then, during that transition, the encoder's output will be 1. If, on the other hand, the transition is indicated by a broken line, the output will be 0. The system thus begins in an arbitrary state, say S_1, and with each clock pulse moves to another state, generating a binary digit in the process. In the diagram of Figure 1 the only way out of S_1 is through a broken line to S_2, thus at least one 0 must follow a 1, which is always the output of the system on its way to S_1. Similarly, S_4 can only lead to S_1 with a solid line, which indicates that no more than three zeros can follow each other without interruption.

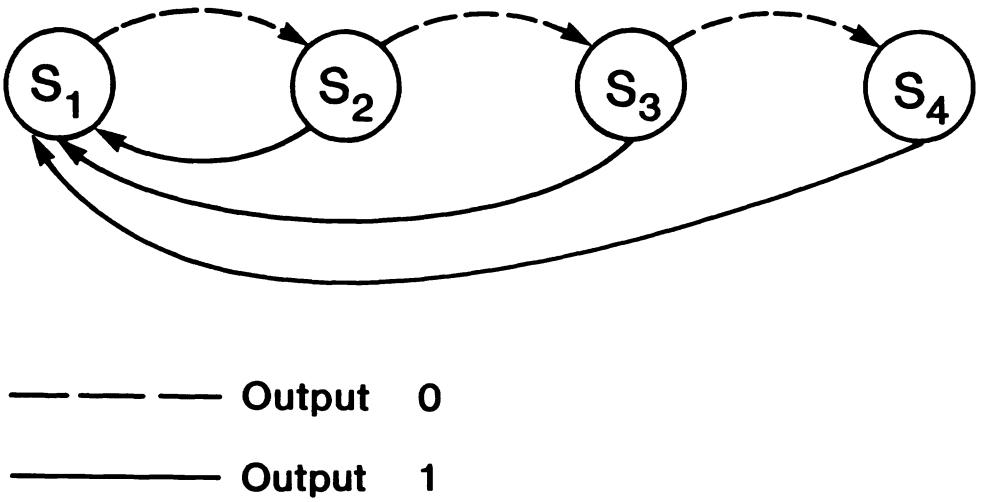

Figure 1. State-transition diagram for the (1,3) code.

Example 2 : The state transition diagram in Figure 2 corresponds to a (2,7) modulation code with one additional constraint. The sequence 1001001001001 is reserved for synchronization and thus cannot appear in the output of the modulation encoder. States S_9 through S_{17} are added to the standard diagram for the (2,7) code in order to accommodate this additional constraint.

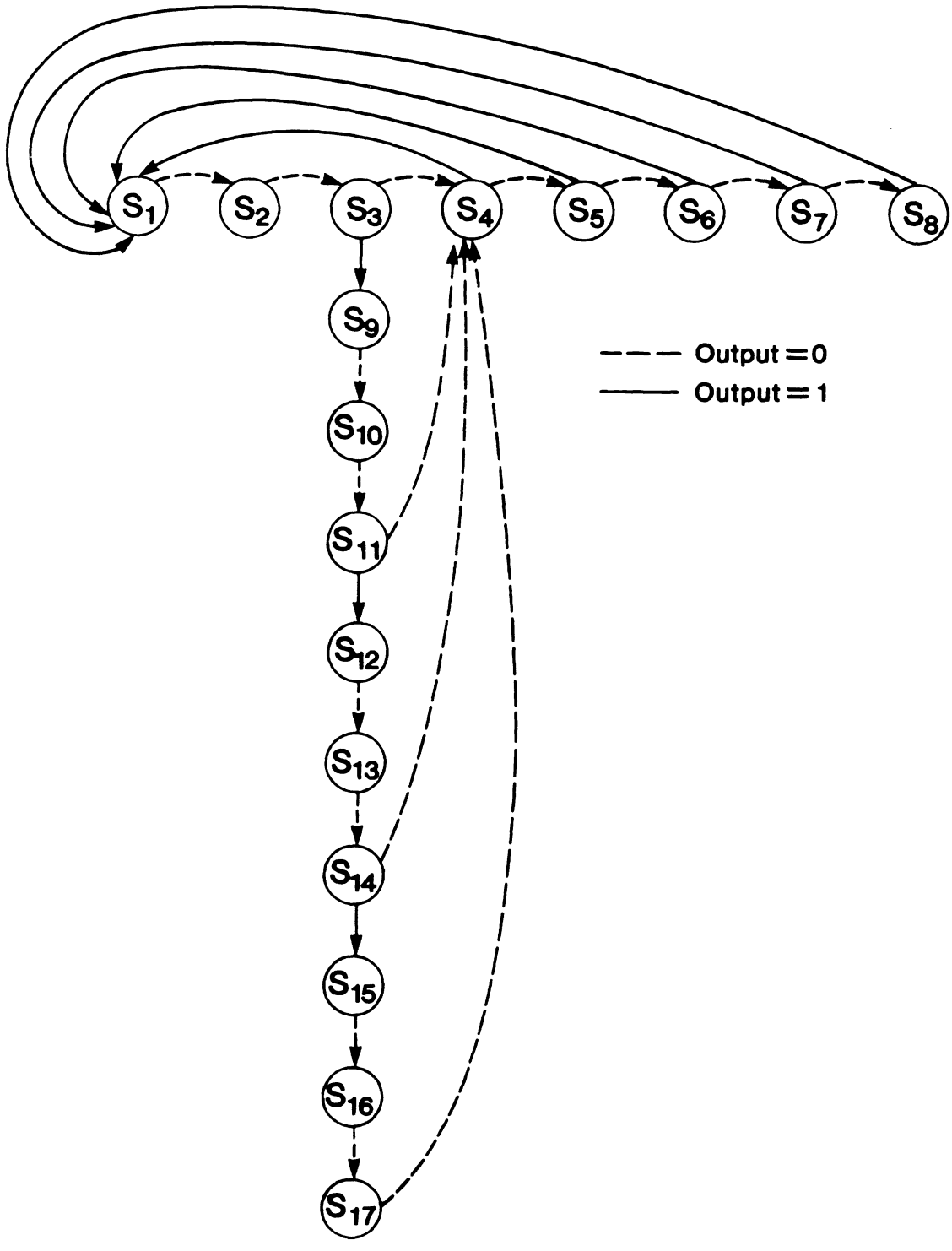

Figure 2. State-transition diagram for the (2,7) code excluding the synchronization sequence 1001001001001.

Example 3 : *A* $(d,k;c)$ code is a (d,k) code with charge constraint c, namely, the accumulated charge in the circuitry during read/write operations must remain in the interval $[-c, +c]$. 1's in the sequence do not change the charge but reverse the direction of its accumulation. For example, if there are n units of charge in the system at a given instant of time and if the charge is on the rise, then immediately after a transition (corresponding to a 1 in the sequence of modulation bits) the charge will still be n units but decreasing. Each 0 in the modulated sequence changes the value of the total accumulated charge by 1 unit in a direction determined by the previous 1's.

Let us consider the case of a (1,3;2) code for which the charge is allowed to be either +2, +1, 0, -1, or -2 units. The state diagram for this code is shown in Figure 3 where $S_m{}^{n\uparrow}$ is defined as the state corresponding to a sequence of m successive 0's after the latest 1, having n units of charge, and with the direction of charge accumulation being "up". If the system begins in $S_0{}^{0\uparrow}$ or $S_0{}^{0\downarrow}$ some states such as $S_0{}^{2\uparrow}$, $S_1{}^{2\downarrow}$, etc. will never be reached. These states are shown as broken circles in Figure 3.

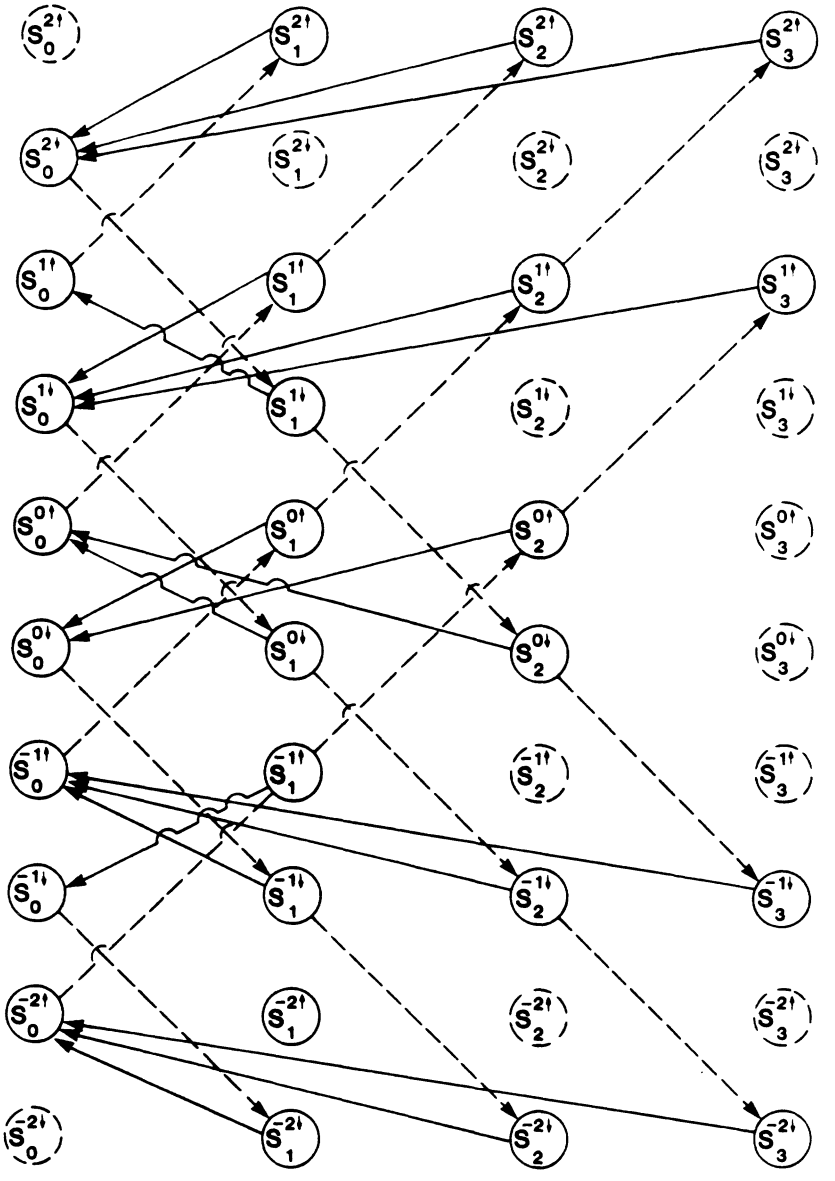

Figure 3. State-transition diagram for the (1,3;2) code

3. THE TRELLIS DIAGRAM

The Trellis diagram is obtained by replicating the state diagram in time. Figure 4 shows the trellis for the state diagram of the (1,3) code discussed in Example 1. Only seven time steps are shown in the Figure; thus this particular diagram corresponds to a modulation code of length $L_0 = 7$. Transitions from a state at time t to a state at time $t+1$ are indicated both with solid lines (representing a 1 output) and broken lines (representing a 0 output). The initial state being chosen as S_1, this is the only state at $t = 0$ that can connect to states at $t = 1$. Like the initial state, the final state can be selected arbitrarily. In fact, one may choose to end in any subset of states (except for the empty subset, of course). In this paper we shall assume that the initial and final states are one and the same in all cases. Accordingly, the final state in Figure 4 is also S_1. This assumption will help simplify the discussion without seriously restricting the scope of the work. It should be emphasized, however, that our particular choice of the final state is by no means essential for the proposed encoding and decoding schemes.

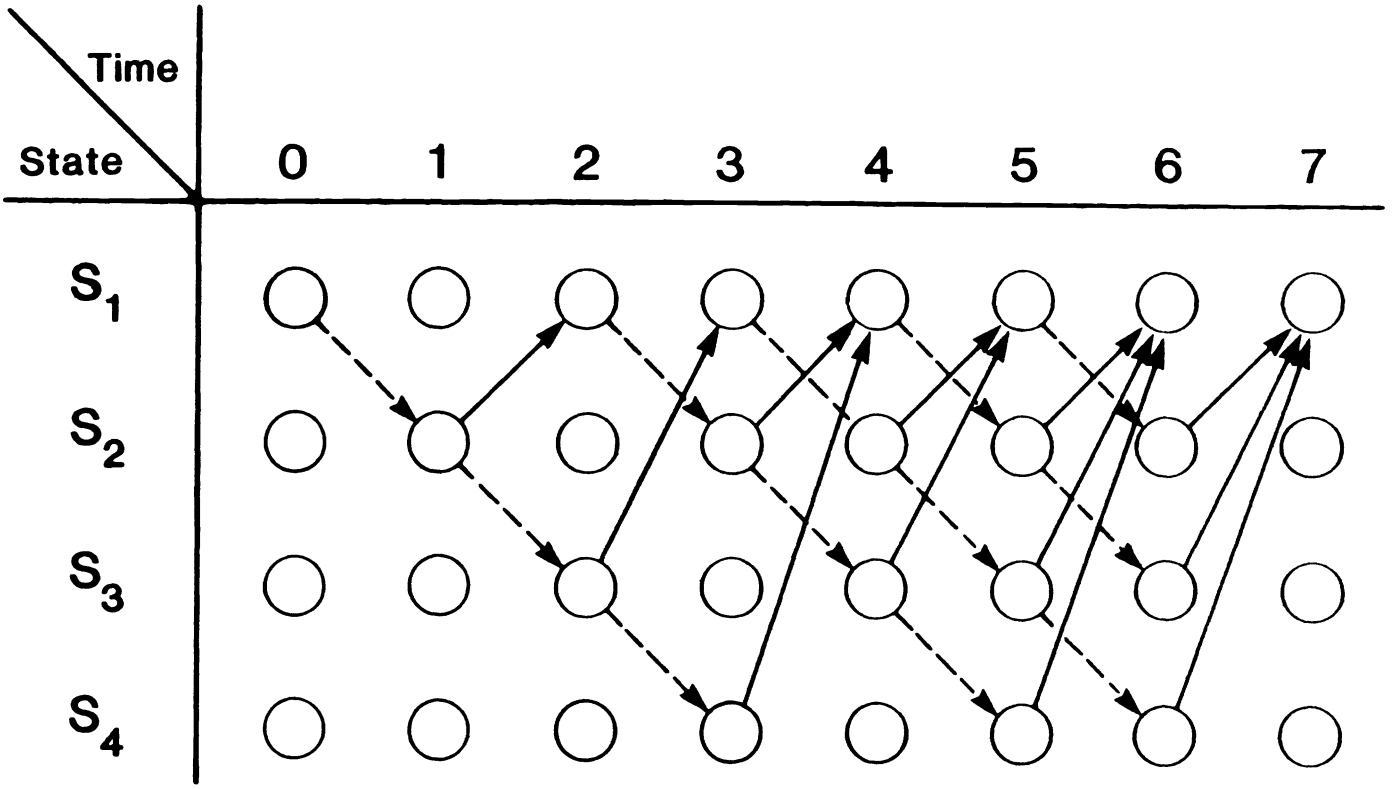

Figure 4. Trellis diagram for the (1,3) code with block length $L_0 = 7$. The initial and final states are both chosen to be S_1.

Once the trellis has been constructed, various code-words of the modulation code may be obtained by simply following a connected path on the trellis between the initial state and the final state. There is a one-to-one correspondence between the connected paths and the acceptable modulation code words. To convert a path into a code word, simply replace each segment of the path with a 1 or a 0, depending on whether that segment is a solid line or a broken line. To find the path corresponding to a given code word, start at the initial state and move to the next state either along a solid line or along a broken line, depending on whether the next digit of the code word is a 1 or a 0. In Figure 4 there are five complete paths, indicating that there are only five code words of length $L_0 = 7$ which satisfy the (1,3) constraint, while beginning and ending in S_1.

We now assign a number to each state S in the trellis. This number shall represent the total number of connected paths between S and the final state. To this end, we begin at the last column of states in the trellis and assign the number 1 to those states that are acceptable as final states. Unacceptable states in this column will receive the number 0. In our example the only acceptable final state is S_1 which, as shown in Figure 5, is the recipient of the number 1 in the last column. Next we move one column to the left and to each state S in this new column, assign the sum of the numbers assigned to the states immediately to the right of S and directly connected to it. By moving to the left one column at a time and repeating the above procedure, we obtain the desired number for every state in the trellis. Note, in particular, that the number associated in this manner with the initial state is the total number of acceptable modulation code words. Figure 5 shows the end result of these assignments for the trellis of Figure 4.

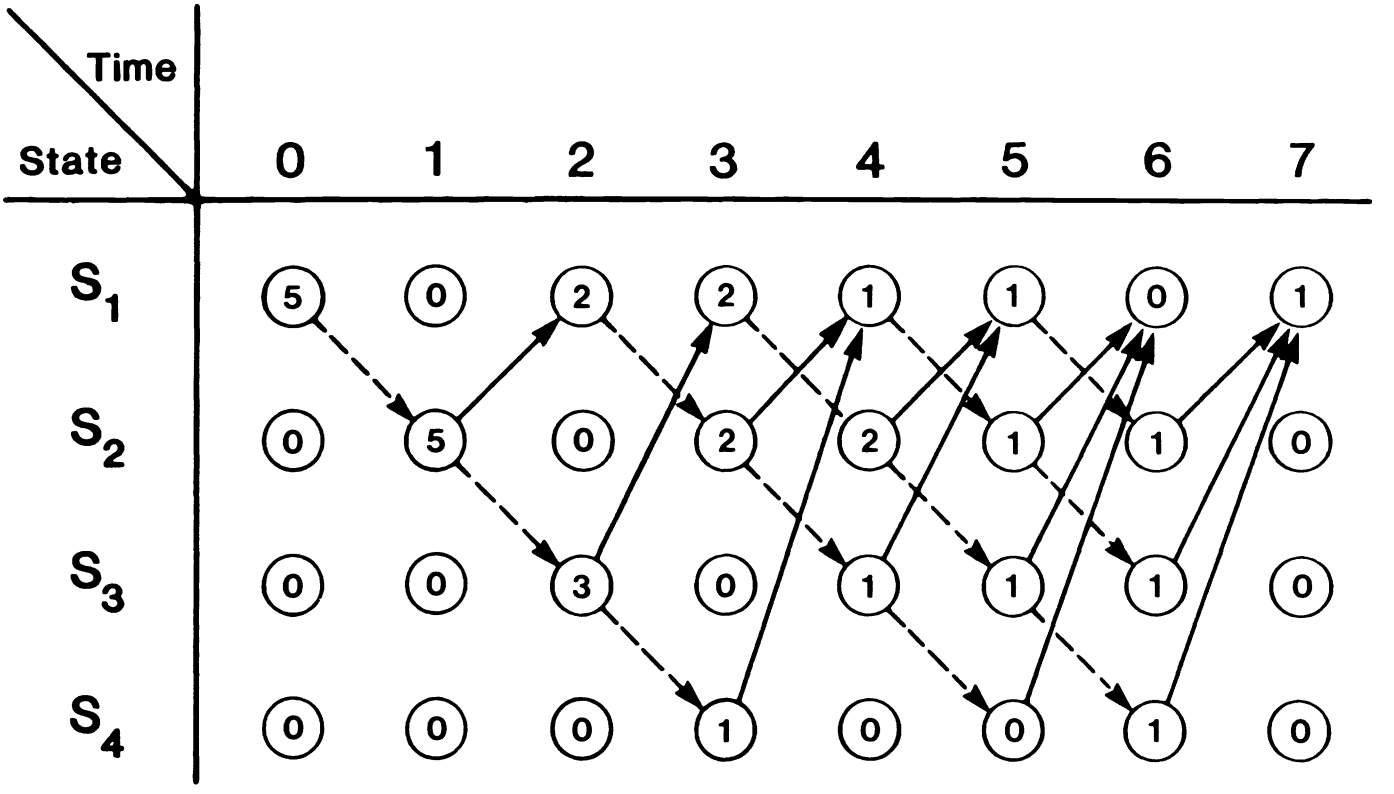

Figure 5. Trellis diagram of Figure 4 with a number assigned to each one of its states. The number assigned to S is the total number of continuous paths between S and the final state.

Next we assign a second number to each state in the trellis. This second set of numbers, which is derived from the first set, will be used in encoding and decoding. In fact the first set will no longer be needed after the construction of the second set has been completed. To obtain the second set we start with the leftmost column of the trellis and assign a number to each state S in that column. The number assigned to S will be 0 if there are less than two lines connecting S to its adjacent states on the right. Thus if a solid line only, or a broken line only, or no lines at all lead from S to the next state, the number assigned to S shall be 0. If, on the other hand, S moves to the next state via *both* a solid line and a broken line, we follow the *broken line* to the corresponding next state, say S', and use the number previously assigned to S' (that is, the total number of connected paths between S' and the final state) as the new number for S. By moving to the right, one column at a time, and repeating the above procedure for all the states in each column, we obtain a complete trellis with new numbers assigned to each state. Figure 6 shows this trellis with new numbers derived from the trellis of Figure 5.

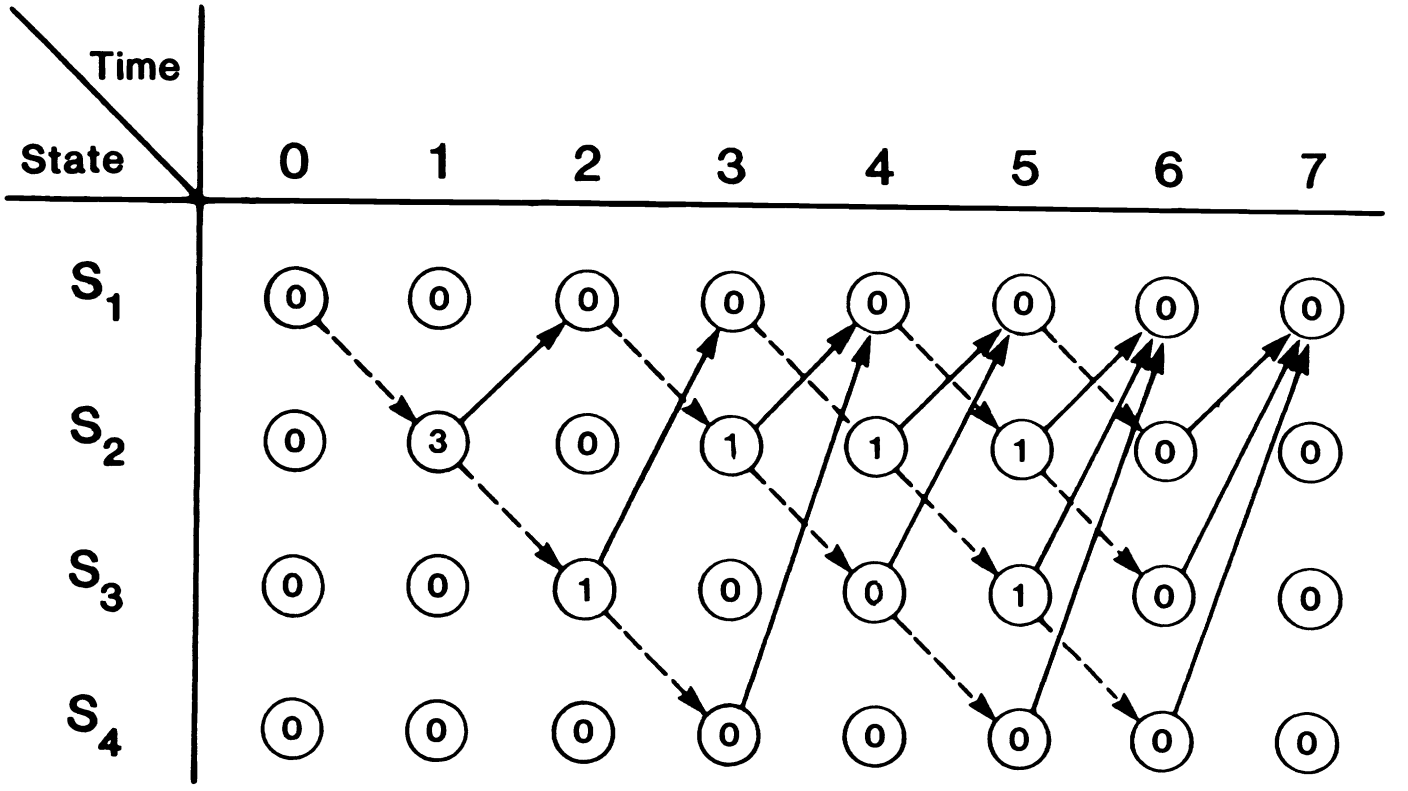

Figure 6. Trellis diagram of Figure 5 with a different number assigned to each one of its states. See the text for a description of the assigned number.

A zero assigned to a state S in the new trellis indicates either that S does not belong to any path that connects the initial state to the final state, or that there is only one path that leads out of S. When a number other than zero is assigned to a state S, it represents the total number of paths that begin at S with a broken line and reach the final state. The set of numbers assigned to the states in this trellis can be stored in an array A of dimensions $L_0 \times \Omega$ where L_0 is the block length of the modulation code words and Ω is the number of states in the state transition table. The array A and the state transition table are all that is needed for modulation encoding and decoding.

4. ENCODING AND DECODING ALGORITHMS

In this section we describe enumerative encoding and decoding algorithms based on the methods described in the preceding section. Consider all possible modulation code-words of length L_0 as obtained, for instance, by identifying all the continuous paths of a given trellis diagram. These code-words may be arranged according to their lexicographic order, just as English words in a dictionary are arranged alphabetically. In fact alphabetical ordering of the code-words is easier since their alphabet consists only of two letters, 0 and 1, and all the words have the same length L_0. Denote the total number of code-words by M_0, and to each code word assign a unique integer (between 0 and M_0 - 1) that represents its position in the above lexicographic order. This is called enumeration and in the following we shall describe a systematic method of obtaining this unique integer for a given code word (i.e., decoding), as well as identifying the code word from the knowledge of its corresponding integer (i.e., reverse enumeration or encoding).

Let us first consider the decoding procedure, since it is slightly easier to describe. With reference to the final trellis diagram discussed in the previous section, a given code-word is simply a connected path between the initial state and the final state. Starting at the initial state and moving along this path, we visit a total of $L_0 + 1$ states. Those states that lead to the next state with a broken line must be ignored, while for the remaining states, (i.e., those that lead to the next state with a solid line), the corresponding numbers on the trellis must be added. The reason being that each time we move along a solid line (which corresponds to a 1 in the code-word), we leap over all the code words that are the same until then but have a zero at that junction. Therefore the lexicographic number of the code-word that follows the solid line must increase by the number of code-words that take the broken line at the junction. The sum thus obtained is the lexicographic number of the code-word under consideration and the decoding is therefore complete. Table I illustrates the decoding process for all the code-words contained in the trellis diagram of Figure 6.

Code-Word	Number
0101001	3 + 1 + 0 = 4
0100101	3 + 0 + 0 = 3
0001001	0 + 0 = 0
0010101	1 + 1 + 0 = 2
0010001	1 + 0 = 1

Table *I*. Code-words corresponding to the trellis of Figure 6 and their respective number in the lexicographic order.

As for the encoding process where the lexicographic order number m ($0 \leq m \leq M_0 - 1$) of a code-word must be translated into the code-word itself, we refer to the same trellis diagram as before, on which we identify the path corresponding to the desired code-word as follows : Define a variable n and set $n = m$ at the outset. Then start at the initial state of the trellis and follow either a broken line or a solid line to the next state : When there is only one line leading to the next state, there is obviously no choice and that line must be followed. However, when there is a choice, compare the current value of n with the number n_s assigned to the current state S of the trellis. If $n \geq n_s$ follow the solid line and replace n with $n - n_s$. If, on the other hand, $n < n_s$ follow the broken line without modifying n. The process continues until the final state is reached, at which point the value of n is zero and the path taken corresponds to the desired code-word.

In using these algorithms for modulation and demodulation of binary user-data it is usually desirable to have a fixed-length block of data mapped onto a fixed-length modulation code-word. If the block length of the user-data is denoted by N_0, the total number of code words M_0 must be greater than or equal to 2^{N_0}. Therefore N_0 must be chosen as the largest integer which is less than or equal to $\log_2 M_0$.

5. BURST-ERROR CORRECTION

There are at least two ways to recognize that a block is affected by a burst of errors. First, when the read signal corresponding to all or a portion of the block has an unusual waveform, that is, when the waveform is atypical of sequences of 0's and 1's. Second, when there is significant and frequent violation of the modulation constraints within the block. These are examples of the so-called erasure channel where, by virtue of the knowledge of the position of error, significant gains in error correction capability of codes may be achieved. In any event, if one or more blocks within a sector are identified as erroneous, it will be possible to restore them with a simple scheme and with a relatively small penalty in overhead. The following example is based on realistic numerical values and explains the proposed method of burst-error-correction.

Suppose that the 512 user Bytes that typically comprise a sector on a disk are divided into 64 blocks of 64 bits each. Subsequently, each block is modulated by some appropriate modulation code and, after additional bits for random error correction and synchronization have been added, we may presume that its length, in units of modulation bits, has become $L = 150$. Suppose now that the longest possible burst that can occur within this sector has length 1050 (in units of modulation bits), and that the probability of two or more bursts occurring within a given sector is negligible. The maximum number of consecutive blocks that can be affected by the burst is therefore eight, which corresponds to 512 successive user bits. The proposed burst-error-correction scheme divides the 512 Bytes of user data (prior to modulation) into eight segments of length 512 bits each, and generates a ninth segment (also of length 512 bits), consisting solely of parity check bits. The n'th parity check bit ($1 \leq n \leq 512$) is chosen such that the n'th bits from all nine segments satisfy the condition of even (or odd) parity. Figure 7 shows the schematic of a circuit that may be used to generate the parity bits in the general case. The total length of the block in this figure is N, the length of each sub-block is K, and CBG stands for Check-Bit-Generator. In the above example, therefore, $N = 4096$ and $K = 512$, resulting in an encoded total length of 4608 bits (user data plus parity bits). It is not difficult now to see that any contiguous group of 512 bits or less (out of this 4608-bit-long block) that is lost to a single burst of errors can be recovered (irrespective of the location of the burst along the block) by virtue of the fact that only one out of every nine bits that originally satisfied the parity conditions are now missing. Thus by adding 512 simple parity bits to the original 4096 bits of the user data, dividing the entire block into 72 sub-blocks of 64 bits each (which sub-blocks are then independently modulated), and storing the resulting 72 modulated blocks all in one sector, we have achieved immunity against single bursts that can be as long as 9.72% of the total length of the sector.

In the above example, the parity bits were generated *before* modulation. A more efficient method of implementing this burst-error-correction technique, however, would generate the parity bits *after* modulation. In this case, because they do not satisfy the modulation constraints, the parity bits must be modulated separately before being recorded (see Section 7 for more on this issue). Also, some intelligence must be built into the decoder in order to enable it to recognize the boundaries of the burst. Merits and demerits of this approach are rather straightforward to fathom; the subject therefore does not warrant an extended discussion in this paper.

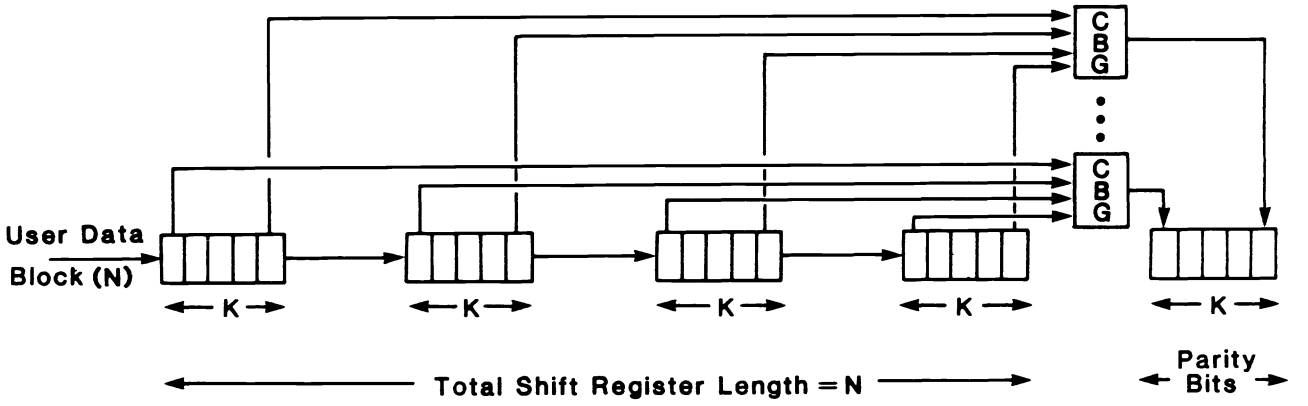

Figure 7. Schematic diagram showing possible implementation of a burst-error-correction encoder. The block of user-data has length N and fills a shift register of the same length. For clarity of representation, this shift register is shown here divided into sub-sections of length K; N and K being chosen such that N is divisible by K. Check-bits are generated by the check-bit generators (CBG's) and stored in another shift register of length K. The input to the n'th CBG comes from the n'th bit of each and every sub-section, and the output comprises the n'th bit of the parity check register. The resulting $(N+K)$-bit-long block (user-data followed by the check-bits) is subsequently sent to the next stage for modulation, where no distinctions will be made between the actual user data and the check bits. In decoding, a contiguous group of erroneous bits (of length K or less), occuring anywhere within this $(N+K)$-bit-long block, can be corrected, provided that the decoder is informed of the location of the burst.

6. VITERBI DECODING

Because modulation constraints have been expressed in the form of state-transition tables, it is rather straightforward to apply Viterbi's algorithm[12-14] during the readout process in order to identify a code-word that not only satisfies all the modulation constraints, but also is closest to the read-back waveform for a given measure of distance (say, in the Hamming sense). Some errors, but not all, can be recovered in this way. The more restrictive the modulation constraints become, the better will be their error-correction capability. In fact, a fruitful subject for future research may be to look for modulation constraints that also allow a significant degree of error-correction without imposing severe penalties on the rate of the code. Some work along these lines has already been reported whereby the advantages of convolutional coding are sought to be combined with those of modulation codes[17]. During numerical experiments we found that $(d,k;c)$ constraints are not powerful enough for the purpose of correcting random errors. The exception was the case of very small c which unfortunately results in small rates.

7. RANDOM-ERROR CORRECTION

The algorithms described in Section 4 are capable of encoding and decoding arbitrarily long blocks of data. The problem, from a practical standpoint, is that a single error in a given block is usually sufficient to cause incorrect decoding of that entire block. This is the well-known problem of error propagation which, in the past, has been "overcome" by choosing codes with short code words, and by preventing the errors from propagating beyond the boundaries of the words within which they occur. A possible solution to the error propagation problem is to apply the error-correcting code to the modulated sequence itself and not, as is common practice today, to the data sequence prior to modulation. In other words, to eliminate the problem of error propagation for codes which have long block lengths, we suggest the use of post-modulation error correction coding. The following example should help clarify the concept.

Let a block of modulation code of length L_0 be error correction coded for a certain number of errors, say ν. If the Reed-Solomon algorithm[15,16] is used for the purpose, it will generate 2ν check-bits (to be appended to the original block of length L_0), but the original block itself will not be modified. These check bits, of course, do not satisfy the modulation constraints and will have to be modulated before appending. There shall be no problems in appending one modulated block to another in this manner, however, since individual blocks always begin and end in the same state. It is even possible to use a different modulation scheme for the check bits; one that is less susceptible to random errors than the modulation scheme used for the user data block. In this case the check bits may not be packed quite as efficiently as the data bits, but they occupy only a small fraction of the total storage area and, therefore, the sacrifice may not be too costly. Of course it is always possible to apply yet another ECC to the block of modulated check bits, and to repeat the process until the integrity of the data after going through modulation and demodulation is assured. Figure 8 shows schematically how a two level post-modulation error correction encoder of this type may be implemented.

8. NUMERICAL RESULTS AND DISCUSSION

A computer program has been developed to generate the state-transition tables for arbitrary (d,k) and $(d,k;c)$ codes, and to implement the proposed enumerative encoding and decoding algorithms. Some of the results obtained with this program are summarized in Tables II and III.

Table II corresponds to three codes with $d = 2$ and $k = 7$. It shows possible block-lengths of the user data (N_0) for several lengths of the code words (L_0). The initial and final conditions for all three codes in this Table are such that each code-word ends in a 1, and begins as though the previous block ended with a 1. Code I is the (2,7) code with no restrictions other than those imposed by the initial and final conditions. The total number of states for this code is 8, and its theoretical maximum rate (Shannon's noiseless capacity) is 0.517[1]. Notice that in Table II the rate increases with increasing L_0, and at $L_0 = 300$ the achievable rate is already equal to 0.510, which is somewhat better than the rate for the currently standard (2,7) code[1]. Code II is similar to code I except that the string 1001001001001 is prohibited from occurring in its code words. The state-transition table for this code, described in Example 2 of Section 2, consists of 17 states. Note that the extra constraint has reduced the code rate by only a small amount. Code III is the (2,7;8) code with the

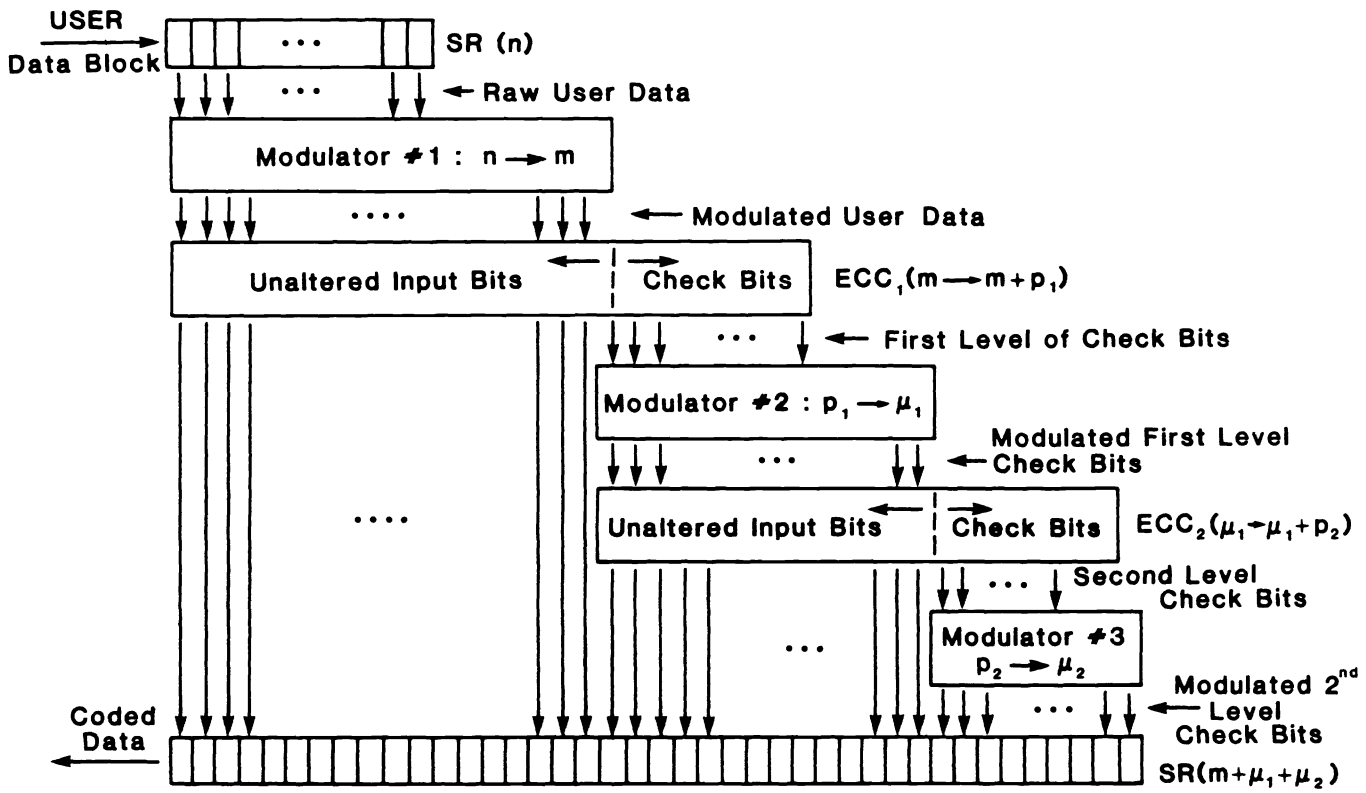

Figure 8. Schematic diagram showing a possible implementation of a two level post-modulation error correction encoder. The block of user-data of length n, placed in a shift register of the same length SR (n), is sent to Modulator #1. Subsequently, the m modulated bits are error correction coded (ECC_1) and p_1 additional bits (the first-level check-bits) are generated. These check-bits are then modulated by Modulator #2 and converted into μ_1 bits. Next, the modulated check-bits are error-correction coded, giving rise to yet another p_2 check-bits at the output of ECC_2. Finally, these second-level check-bits are modulated and yield μ_2 bits at the output of Modulator #3. The final code-word in this example consists of the m modulated user-bits, followed by μ_1 modulated first-level check-bits, followed by μ_2 modulated second-level check-bits.

terminal states chosen such that, in addition to satisfying the previous boundary conditions, each code-word begins and ends with zero charge. The total number of states for this code is 210, and its theoretical maximum rate is 0.501[1]. As can be seen in Table II, the achievable rate increases with increasing L_0, and for $L_0 = 300$ the rate of the code is 0.483.

Table III corresponds to two codes which both have $d = 1$ and $k = 6$. Again the initial and final conditions are such that each code word ends in a 1, and begins as though the previous block ended with a 1. Code IV is the (1,6) code with no restrictions other than those imposed by the initial and final conditions. The total number of states for this code is 7, and its theoretical maximum rate is 0.669[1]. Notice that at $L_0 = 200$ the achievable rate is already equal to 0.660. Code V is the (1,6;8) code which, like code III, begins and ends in a state with zero charge. The total number of states for this code is 194. Clearly, the achievable rate increases with increasing L_0 and, for $L_0 = 300$, the rate of the code is 0.637.

Modulation Block-Length (L_0)	User-Data Block-Length (N_0) Code *I*	Code II	Code III
50	23	22	19
100	49	48	44
150	75	74	70
200	101	100	95
250	127	125	120
300	153	151	145

Table II. Comparison among three codes with $d = 2$ and $k = 7$ for different code-word lengths L_0. Code I is the (2,7) code with no restrictions other than those imposed by the initial and final conditions. Code II is the same as code I except that the string 1001001001001 is prohibited from occurring in the code-words. Code III is the (2,7;8) code with each code-word beginning and ending with zero charge. The initial and final conditions for all three codes are such that each code word ends in a 1 and starts as though the previous block ended with a 1.

Modulation Block-Length (L_0)	User-Data Block-Length (N_0) Code IV	Code *V*
50	31	27
100	65	60
150	98	93
200	132	125
250	165	158
300	198	191

Table III. Comparison between two codes with $d = 1$ and $k = 6$ for different code-word lengths L_0. Code IV is the (1,6) code with no restrictions other than those imposed by the initial and final conditions. Code *V* is the (1,6;8) code with each code-word beginning and ending with zero charge. The initial and final conditions for both codes are such that each code-word ends in a 1 and starts as though the previous block ended with a 1.

9. CONCLUDING REMARKS

In this paper we have described methods and algorithms for the modulation and error-correction-coding of unconstrained streams of data. The intended application of these ideas is in the general area of magnetic and optical data storage. In the course of this exposition we have avoided abstractions and rigorous mathematical statements in order to convey the beauty and the power of the concepts to the broadest possible audience.

The examples given so far have concentrated on known classes of modulation codes, but the proposed techniques are much more powerful than their predecessors and should be applied to new problems and in new directions. For instance, a possible approach to the optimal design of constrained sequences for a given data

storage system might entail the experimental determination of curves similar to those shown in Figure 9. Given a particular storage device and for a given clock cycle T, one must experiment with pulses whose durations are integer multiples of T. Recording and reading these pulses under realistic conditions, one obtains, after a fair number of trials, upper and lower bound curves for the length of the readout pulse, as shown in Figure 9(a). Run-lengths which do not overlap each other after being recorded and then retrieved, are readily identifiable from these curves, as shown in Figure 9(b). The correct set of constraints for such a system is now given by the collection of these non-overlapping run-lengths. For the imaginary system characterized by the curves of Figure 9 the acceptable run-lengths are 4, 8, 12, 20, ···. Once an acceptable set of run-lengths has been identified the state transition table can be constructed and the enumerative encoder and decoder implemented. The choice of maximum run-length is dictated by the acceptable level of complexity for the encoder and decoder. In fact the choice of the clock cycle T is not arbitrary either, and one might try to optimize it by trial and error, keeping in mind that although smaller values of T result in higher densities, the complexity of implementation grows with decreasing clock cycle.

The ideas of enumerative modulation coding and post-modulation error correction described in this paper have removed some of the barriers to achieving high density storage by allowing the potential of the media and systems to be fully realized. It is hoped that this exposition will facilitate the transfer of these ideas to practice.

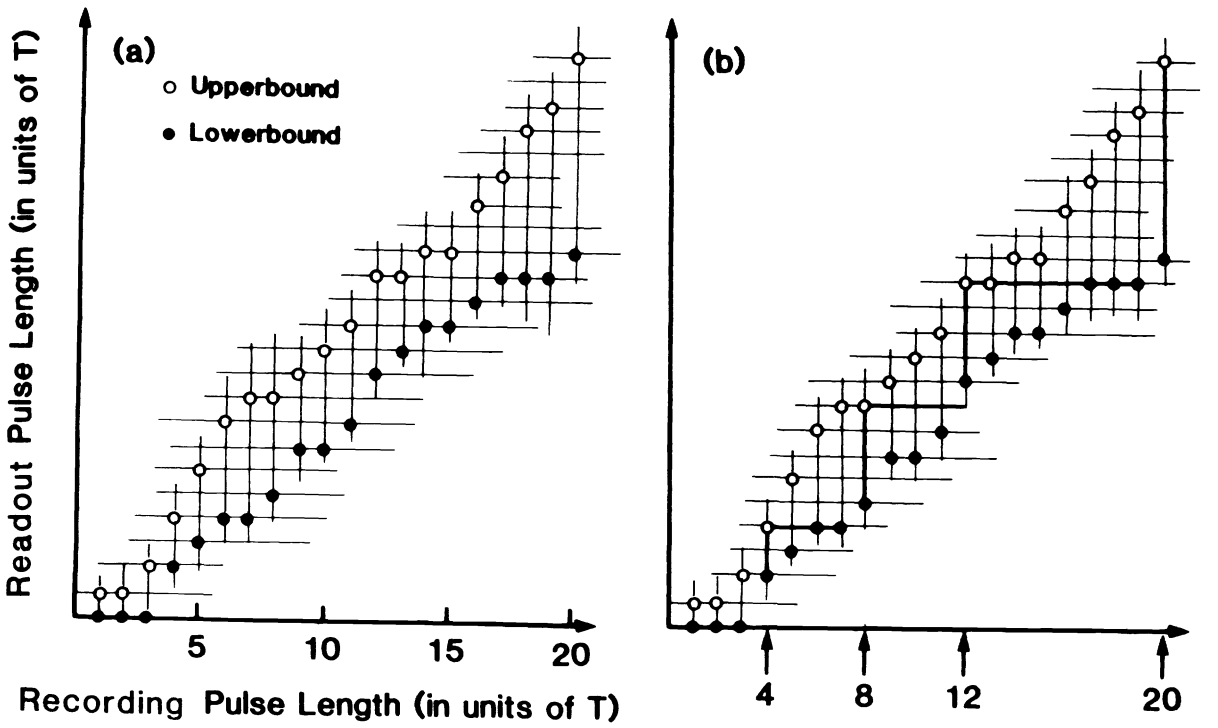

Figure 9. (a) Presumed upper and lower bounds on the lengths of various recorded pulses after readback. The horizontal axis represents the length of the pulse before recording. The durations of both read and write pulses are multiples of the clock cycle T. These curves can be obtained for a given system by repeatedly recording marks of various lengths under realistic conditions and monitoring the corresponding signal in readout. (b) Method of selecting run-lengths for modulation. The first accepted run-length is the shortest pulse for which the lower bound is non-zero. Other run-lengths have the shortest possible length that allows them to avoid overlap.

ACKNOWLEDGEMENTS

The ideas presented in this paper originated during many hours of discussion between the author and Dr. Boris Fitingof who was a visiting scholar at the Optical Data Storage Center (University of Arizona) from April 1989 until June 1990. I thank Dr. Fitingof for sharing of his ideas. I also thank Dr. Dennis Howe of Kodak Research Laboratories, Professor Lev Levitin of Boston University, and Dr. Arvind Patel of IBM's General Products Division for helpful discussions and constructive criticism.

REFERENCES

1. A. M. Patel, *Signal and Error Control Coding*, in **Magnetic Recording, Volume II**, Eds., C. D. Mee and E. D. Daniel, McGraw-Hill, New York, 1988.

2. P. A. Franaszek, *Run-Length Limited Variable Length Coding with Error Propagation Limitation*, U.S. Patent 3,689,899. 1972.

3. D. G. Howe, *The Nature of Intrinsic Error Rates in High-Density Digital Optical Recording*, in Proceedings of SPIE, Vol. 421, **Optical Disk Systems and Applications**, *pp.*31-42 (1983).

4. D. G. Howe, *Signal to Noise Ratio (SNR) for Reliable Data Recording*, in Proceedings of SPIE, Vol. 695, **Optical Mass Data Storage II**, *pp.*255-261 (1986).

5. K. S. Immink, *Coding Methods for High-Density Optical Recording*, Philips J. Res. **41**, *pp.*410-430 (1986).

6. R. L. Adler, D. Coppersmith and M. Hassner, *Algorithms for Sliding Block Codes*, IEEE Trans. Inform. **29**, *pp.*5-22 (1983).

7. A. M. Patel, *Zero-Modulation Encoding in Magnetic Recording*, IBM J. Res. Dev. **19**, *pp.*366-378 (1975).

8. D. T. Tang and L. R. Bahl, *Block Codes for a Class of Constrained Noiseless Channels*, Information and Control **17**, 436-461 (1970).

9. C. E. Shannon, *A Mathematical Theory of Communication*, Bell Syst. Tech. J., **27**, 379 (1948).

10. B. Fitingof and Z. Waksman, *Fused Trees and Some New Approaches to Source Coding*, IEEE Trans. Inform. Theory, Vol. IT-34, pp.417-424, May 1988.

11. T. Cover, *Enumerative Source Coding*, IEEE Trans. Inform. Theory, Vol. IT-19, *pp.*73-76 Jan. 1973.

12. A. J. Viterbi and J. K. Omura, **Principles of Digital Communication and Coding**, McGraw-Hill, New York, 1979.

13. G. D. Forney, *Convolutional Codes II: Maximum Likelihood Decoding and Convolutional Codes III: Sequential Decoding*, Infor. and Contr., **25**, pp.222-297 (1974).

14. M. Mansuripur, **Introduction to Information Theory**, Prentice-Hall, New Jersey, 1987.

15. I. S. Reed and G. Solomon, *Polynomial Codes over Certain Finite Fields*, J. SIAM, **8**, pp.300-304 (1960).

16. R. E. Blahut, **Theory and Practice of Error Control Codes**, Addison-Wesley, Reading, MA (1983).

17. J. K. Wolf and G. Ungerboeck, *Trellis Coding for Partial-Response Channels*, IEEE Trans. Comm., Vol. COM-34, *pp.*765-773 (1986).