

# A Practical Method for Approaching the Channel Capacity of Constrained Channels

Kees A. Schouhamer Immink, *Fellow, IEEE*

**Abstract**—A new coding technique is proposed that translates user information into a constrained sequence using very long codewords. Huge error propagation resulting from the use of long codewords is avoided by reversing the conventional hierarchy of the error control code and the constrained code. The new technique is exemplified by focusing on  $(d, k)$ -constrained codes. A storage-effective enumerative encoding scheme is proposed for translating user data into long  $dk$  sequences and *vice versa*. For  $dk$  runlength-limited codes, estimates are given of the relationship between coding efficiency versus encoder and decoder complexity. We will show that for most common  $d, k$  values, a code rate of less than 0.5% below channel capacity can be obtained by using hardware mainly consisting of a ROM lookup table of size 1 kbyte. For selected values of  $d$  and  $k$ , the size of the lookup table is much smaller. The paper is concluded by an illustrative numerical example of a rate  $256/466$ , ( $d = 2, k = 15$ ) code, which provides a serviceable 10% increase in rate with respect to its traditional rate  $1/2, (2, 7)$  counterpart.

**Index Terms**—Constrained code, enumerative coding, recording code, runlength-limited.

## I. INTRODUCTION

CONSTRAINED codes have been used extensively in recording products and in data transmission. In a state-of-the-art recorder (see Fig. 1) the channel encoder translates source data plus error-correction (parity check) symbols into a sequence that complies with the channel constraints imposed.

In virtually all recording systems in the market, the error correction code (ECC) is based on Reed–Solomon codes [1]. There are, however, many flavors in channel codes. Notably, spectral shaping and runlength-limited codes have found widespread usage in consumer-type recording systems such as Compact Disc, DAT, DVC, etc. [2]. Since only a limited repertoire of sequences is used, the rate of a code that transforms input data into a constrained sequence is by necessity less than unity. The maximum information rate given the channel input constraints is called the *Shannon capacity* of the input-constrained noiseless channel. A good channel code embodiment realizes a code rate that is close to the Shannon capacity of the constrained channel, uses a simple implementation, and avoids the propagation of errors at the decoding process, or, more realistically, a code with a compromise between these competing attributes.

Manuscript received September 30, 1996; revised January 20, 1997. The material in this paper was presented in part at ISITA'96, Victoria, Canada, September 18–20, 1996.

The author is with Philips Research Laboratories, 5656 AA Eindhoven, The Netherlands.

Publisher Item Identifier S 0018-9448(97)05714-3.

For reasons of hardware and error propagation [3], current channel code implementations are very often small and byte-oriented. The efficiency of such small codes, in terms of channel capacity, is typically less than 97% [2]. In accordance with the adage “The bigger the better,” a greater code efficiency can only be realized by codes with codewords that are much longer than currently in use. Although we know how to construct such efficient codes in theory, the key obstacle to practically approaching channel capacity is the massive hardware required for encoding and decoding, as hardware grows with the number of codewords used, i.e., exponentially with the codeword length. The encoding and decoding massiveness can be solved by using a technique called *enumeration* [4]. The enumerative coding technique makes it possible to translate source words into codewords and *vice versa* by invoking an algorithmic procedure rather than performing the translation with a lookup table. The second drawback of the use of long codewords, which still remains, namely, the risk of extreme error propagation, precluded its usage in practical systems. Single channel bit errors may result in error propagation which could corrupt the entire data in the decoded word, and, of course, the longer the codeword the greater the number of data symbols affected.

The new technique to be discussed below offers a step-by-step method to solve the practical problems pertaining to the usage of long constrained codewords. As a result, the new technique makes it possible to employ codes having very large codewords thereby approaching the capacity of the constrained channel. We start with a description of the new coding format in relation to error correction coding and we propose a new technique for avoiding massive error propagation. We envision that this technique can be used in conjunction with any constrained code, but in this paper we have chosen to exemplify the new technique by focusing on  $(d, k)$ -constrained codes. To that end, we briefly outline the relevant prior art of  $(d, k)$ -constrained block codes, followed in Section IV with a description of a new and storage-effective enumeration technique of  $(d, k)$ -constrained codes. In Section V, we will give special attention to the relationship between the hardware requirements and the efficiency of long  $(d, k)$  codes. Two new codes are given to illustrate the new construction method.

## II. FIGURE 1 REVISITED

As suggested by Bliss [5], and recently Mansuripur [6], we propose to solve the error propagation problem inherent in the usage of long channel codewords by reversing the normal

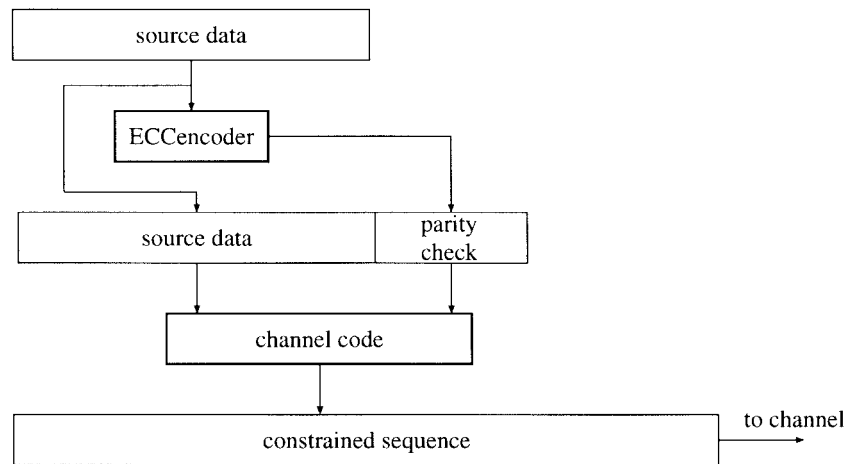


Fig. 1. Block diagram of the two coding steps in a recording system.

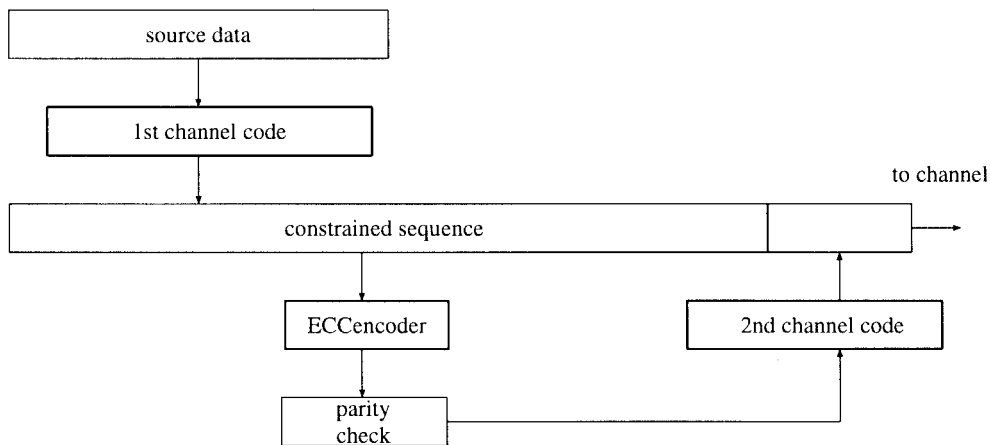


Fig. 2. Code configuration of two-step coding, "Bliss scheme."

conventional hierarchy of the error correction and channel codes shown in Fig. 1. A block diagram of Bliss' encoding scheme is shown in Fig. 2.

In Bliss' coding hierarchy, a first channel encoder translates blocks of user data into codewords that obey the prescribed channel constraints. The constrained codewords, in turn, are treated as binary input data of an error-correcting code in the usual fashion. In a byte-oriented error control code, such as the popular Reed–Solomon (RS) code, the constrained codewords are partitioned into bytes and the parity bytes generated are affixed to the end (or beginning) of the constrained codeword. The parity bytes generated do not, in general, obey the prescribed constraints and must therefore be modified by a second channel code. Note that provisions have to be made so as to be able to efficiently concatenate the coded segments generated by the first and second channel code.

At the receiver's site, we start with the decoding of the parity bytes. Subsequently, following the rules of the ECC at hand, we correct possible errors in the parity words or the first channel codeword. Thereafter, we decode the first channel code. Error propagation during decoding of the first channel code is irrelevant as it is assumed that all transmission errors made can be corrected by the ECC. This property opens the

door to using codes with long codewords without the risk, as in the Fig. 1 configuration, of extreme error propagation. The quintessence of Bliss' scheme is that the first channel code can be made very efficient as it uses very long codewords. The second channel code may be any readily available code having the (low) efficiency of standard constrained codes. As the number of parity bits is normally a small fraction of the number of input bits, the lower efficiency of the second channel code has a relatively small bearing on the overall efficiency. Note that it is important that the error propagation of the second channel code is limited to a few bits, preferably to a single byte in a byte-oriented system. In principle, simple codes developed with the aid of the state-splitting algorithm can be used. There are two flaws in Bliss' scheme making it unattractive in certain applications.

In Bliss' scheme, the ECC code has the constrained sequence as an input. As the constrained sequence is by necessity redundant, we do not fully exploit the capability of the ECC. As a result, we may discern two difficulties. Assume the ECC can correct bursts of errors of at most  $t$  symbols. Then, assuming a  $t$  error burst occurs in the block generated by the first channel code, the ECC is able to correct the constrained sequence of  $t$  symbols. This corresponds with an

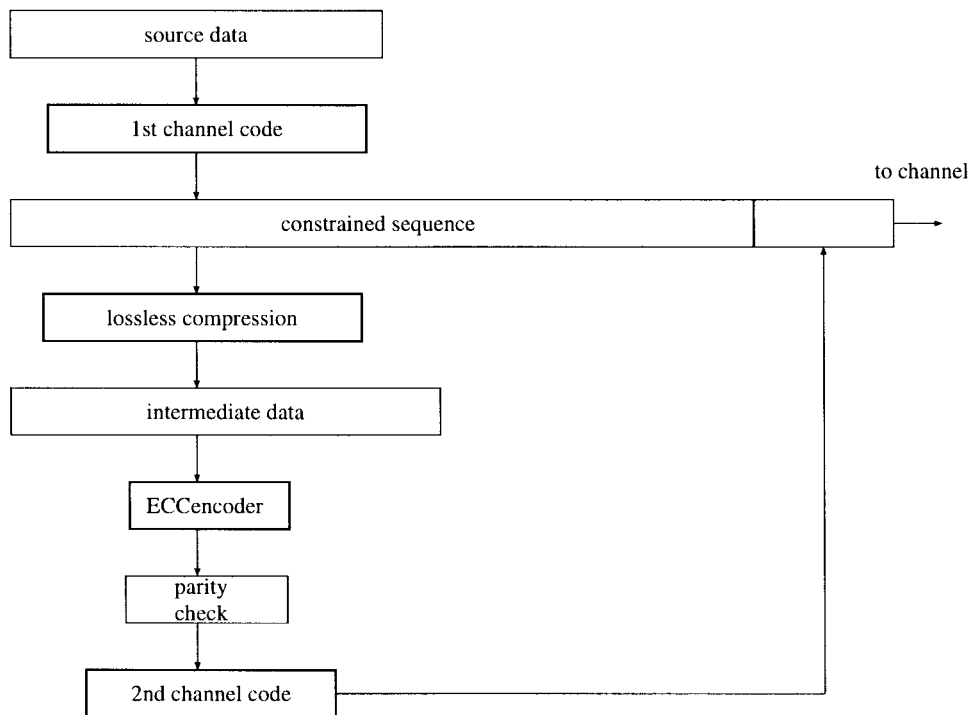


Fig. 3. New code configuration.

error burst of length approximately  $R_1 t$  user symbols, where  $R_1$  is the rate of the first constrained code. As the same ECC would have corrected  $t$  user symbols in the conventional code configuration, we conclude that in Bliss' scheme a loss by a factor of  $R_1$  in maximum burst error correcting capability is incurred. Secondly, in Bliss' scheme the ECC faces an input sequence whose length is expanded by a factor of  $1/R_1$  with respect to the equivalent one in the conventional, "Fig. 1," coding scheme. As the length of the input sequence must be smaller than the maximum imposed by the (RS) code at hand, we may sacrifice design freedom, which could result in loss of code performance. An analysis of Bliss' scheme is presented by Fan and Calderbank [7].

The above drawbacks of Bliss' scheme can be solved by reconfiguring the codes and defining a third intermediate coding layer. Essentially, in the new configuration the redundancy of the constrained sequence is removed by a lossless compression step before the sequence is forwarded to the ECC code. By definition, the most efficient lossless compression can be done by the first channel decoder (and retrieving the source data again), but this pointless case has the drawback of error propagation. Therefore, we apply a less efficient lossless compression with limited error propagation. This can, for example, be accomplished with the aid of a fixed one-to-one block mapping. Fig. 3 shows a block diagram of the new encoding configuration. The constrained sequence is partitioned into blocks of  $q$  bits before it is forwarded to the ECC. The block length  $q$  is chosen as large as possible on the understanding that the number of distinct constrained sequences of length  $q$  is not larger than the field size of the symbol error-correcting code  $N$ . In practice this is usually a byte-oriented code, thus  $N = 256$ . It is therefore possible

to define a one-to-one mapping between the  $q$ -tuples and the ECC symbols. Using a small lookup table  $\mathcal{L}$ , or enumeration, the  $q$ -tuples can be unambiguously translated into a sequence of bytes which, in turn, is used as the input of the RS encoder. As in the Bliss' scheme, the parity check bytes are translated by a second channel code. The constrained sequences generated by the first and second channel codes are cascaded and transmitted. As a result of the compression operation, we may expect that the number of bytes entering the ECC is smaller than the number of bytes that enter the ECC in the original Bliss' scheme, but larger than the original number of source bytes. Let  $R_c$  denote the rate of the compression stage, then the expansion factor is  $R_c/R_1$ , namely, the quotient of the lengths of the compressed sequence and the source sequence. The effectiveness of the compression scheme depends on the channel constraints at hand and the particular choices of the length of the codewords. Two worked examples of constrained codes plus the lossless compression schemes and their efficiency are presented in Section VI-A. In the two examples shown, the expansion factor is 1.06 and 1.13, respectively.

At the receiver's site, the retrieved sequence is decoded in a few steps (see Fig. 4). First, the parity symbols are simply found under the decoding rules of the second channel decoder. Then, the first constrained sequence is chopped into  $q$ -tuples, which using the lookup table  $\mathcal{L}$  are translated into an intermediate sequence of ECC symbols, say bytes. The intermediate sequence and the parity symbols may contain transmission errors, which can be corrected with the aid of the ECC. After this ECC decoding operation, the corrected intermediate sequence of symbols is translated again into a constrained sequence using the inverse of the lookup  $\mathcal{L}$ . At

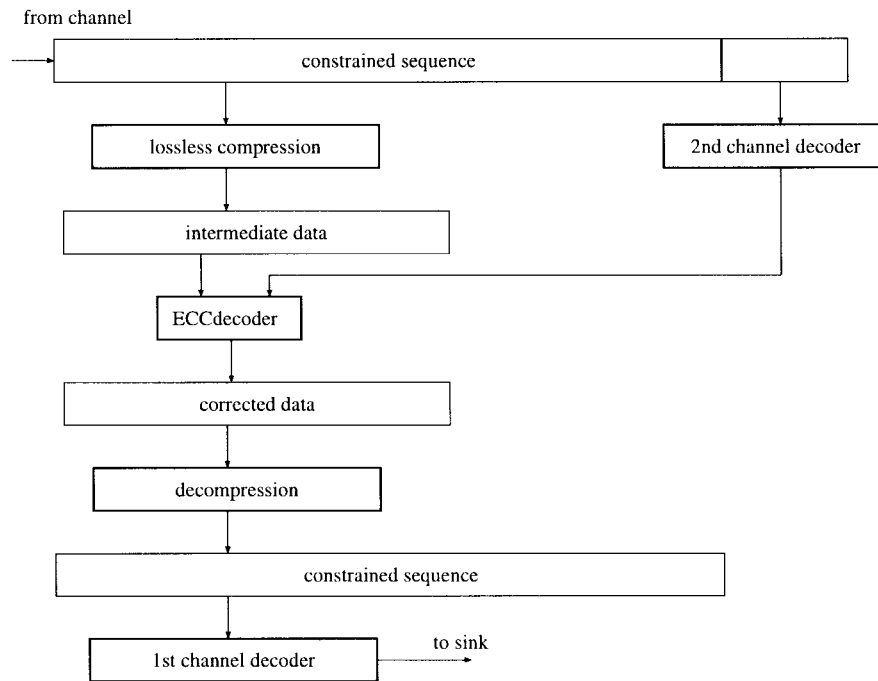


Fig. 4. Decoder configuration of the new coding scheme.

this level, it can reasonably be assumed that the corrected constrained sequence is error-free. Next, decoding of this sequence takes place using the first channel decoder, and, finally, the retrieved message can be delivered to the recipient. Note that the additional translation into the constrained sequence using the inverse of the lookup  $\mathcal{L}$  makes it possible to use the decoding algorithm.

We could contemplate reinvesting (part of) the profit realized by using an efficient recording code in an improvement of the burst error correcting capability of the ECC. The various tradeoffs are the province of the system architect and are thus beyond the scope of this paper.

The new coding configuration can be used in conjunction with channels having any type of channel constraint. We will by way of example illustrate the effectiveness of the new technique by focusing on  $(dk)$ -constrained codes. As an introduction to this matter, the next section will provide a brief overview of relevant constructions of  $(dk)$ -constrained block codes.

### III. LONG $(dk)$ -CONSTRAINED BLOCK CODES

A  $dk$ -limited (binary) sequence,  $(dk)$  sequence for short, simultaneously satisfies the following two conditions: 1)  $d$  constraint—two logical “ones” are separated by a run of consecutive “zeros” of length at least  $d$  and 2)  $k$  constraint—any run of consecutive “zeros” is of length at most  $k$ . If only condition 1) is satisfied, the sequence is said to be  $d$ -limited (with  $k = \infty$ ), and will be termed  $d$ -sequence. Similarly, if only condition 2) is satisfied the sequence is said to be  $k$ -limited.

Below we will show that the systematic design of long  $dk$  constrained codes is feasible with block codes. It is obvious that when the codeword length is large, the encoder/decoder mapping should be such that they can be implemented by an

algorithmic procedure rather than by table lookup. Block-code constructions that can be used in conjunction with such an algorithm have been presented by Tang and Bahl [8], Beenker and Immink [9], Gu and Fuja [10], and Immink [11]. A survey and comparison of the various block-code constructions have been given by Abdel-Ghaffar and Weber [12]. Gu and Fuja’s method is optimal: no other block code can achieve a higher rate. Recently, Immink [11] presented a code construction which is “almost” block-decodable. Gu and Fuja [10] showed that Beenker and Immink’s “Construction 2” is optimal for  $d = 0$  and  $d = 1$ . Construction 2 is not optimal for  $d \geq 2$ . Construction 2, though not optimal for all  $(d, k)$  parameters, has the advantage that only *one* table is needed plus simple logic to cascade the sequences. Below we will briefly discuss the various block-code constructions. To be able to do so we need a definition.

*Definition:* A  $(dklr)$  sequence is a  $(dk)$  sequence with additional constraints on the number of leading and trailing zeros:

- 1)  $l$  constraint—the number of consecutive leading “zeros” of the sequence, that is, the number of “zeros” preceding the first “one,” is at most  $l$ .
- 2)  $r$  constraint—the number of consecutive trailing “zeros” of the sequence, that is, the number of “zeros” succeeding the last “one,” is at most  $r$ .

Finite-length  $(dklr)$  sequences cannot in general be cascaded without offending the given  $(dk)$  constraints at the boundaries. Inserting a number  $\beta$  of “merging” bits between adjoining sequences makes it possible to preserve the  $d$  and  $k$  constraints. Cascading (merging) can be done with the following constructions [9].

*Construction 1:* Choose the parameters  $d, k, r$ , and  $l$  such that  $r + l \leq k - d$ , and let the number of merging bits be  $\beta = d$ .

Every message is associated with a  $(dklr)$  sequence of length  $n - d$ . As a direct result of the judicious choice of  $l$  and  $r$ , the  $(dklr)$  sequences can be *freely* cascaded without violating the specified  $d$  and  $k$  constraints, provided the  $\beta$  merging bits are all set to “zero.” The number of  $(dklr)$  sequences is maximized if  $l$  and  $r$  are  $l = \lfloor (k - \beta)/2 \rfloor$  and  $r = k - \beta - l$ , where  $\lfloor x \rfloor$  means the largest integer in a nonnegative number  $x$ .  $\square$

*Construction 2:* Choose the parameters  $d, k$ , and  $n$  such that  $k \geq 2d$  and  $d \geq 1$ . Let  $r = l = k - d$  and  $\beta = d$ . Then the  $(dklr)$  sequences can be cascaded without violating the specified  $(d, k)$  constraints if the  $\beta$  merging bits are judiciously chosen [9].  $\square$

*Construction 3:* Choose the parameters  $d, d \geq 2, k$ , and  $n$  and let the number of merging bits be  $\beta = d - 1$ . We have to choose  $k$  so that  $k - \lfloor d/2 \rfloor > d$ . The parameters  $r$  and  $l$  are specified by  $l + r = k - \beta$ . The set of permitted codewords is maximized if  $r = \lfloor (k - \beta)/2 \rfloor$  and  $l = (k - \beta - r)$ . Then the  $(d, k - \lfloor d/2 \rfloor, l, r)$  sequences of length  $n - \beta$ , excluding the words  $0^{n-\beta}$  and  $0^{n-\beta-1}1$ , can be cascaded with a simple rule [11].  $\square$

Codes constructed with Constructions 1 and 2 are block-decodable as the original  $(dklr)$  sequences remain unchanged during the merging process. Decoding is done by observing the  $n - \beta$  bits of the codewords that have a one-to-one relationship with the source words. The  $\beta$  merging bits do not contain relevant information for the decoder, and are therefore skipped. In contrast to both Constructions 1 and 2, decoding is done in Construction 3 by observing  $n - \beta$  bits of the received word *plus* the pivot bits preceding and succeeding the actual codeword.

In the next section, we will discuss an algorithm for encoding and decoding  $(dklr)$  sequences. The algorithm, called enumeration, is an essential part of the new coding method.

#### IV. CODING AND DECODING USING ENUMERATION

The translation of input data into constrained sequences and *vice versa* using enumeration has a long history [13]. The first encoder using enumeration was patented by Labin and Aigrain [14] in 1951. The inventors described a mechanism for translating user data into 35 binary codewords each having 3 out of 7 “ones.” It is not known if this has been put to any practical use. In 1965, Kautz [15] presented an enumeration scheme for coding  $(0, k)$ -constrained sequences, followed, in 1970, by a general enumeration scheme of  $(dk)$ -constrained sequences by Tang and Bahl [8]. The description of the enumeration method given below is due to Cover [4].

Let  $\{0, 1\}^n$  denote the set of binary sequences of length  $n$  and let  $S$  be any (constrained) subset of  $\{0, 1\}^n$ . The set  $S$  can be ordered lexicographically as follows: if  $\mathbf{x} = (x_1, \dots, x_n) \in S$  and  $\mathbf{y} = (y_1, \dots, y_n) \in S$ , then  $\mathbf{y}$  is called less than  $\mathbf{x}$ , in short,  $\mathbf{y} < \mathbf{x}$ , if there exists an  $i$ ,  $1 \leq i \leq n$ , such that  $y_i < x_i$  and  $x_j = y_j$ ,  $1 \leq j < i$ . For example, “00101”  $<$  “01010.” The position of  $\mathbf{x}$  in the lexicographical ordering of  $S$  is defined to

be the *rank* of  $\mathbf{x}$ , denoted by  $i_S(\mathbf{x})$ , i.e.,  $i_S(\mathbf{x})$  is the number of all  $\mathbf{y}$  in  $S$  with  $\mathbf{y} < \mathbf{x}$ .

Let  $n_s(x_1, x_2, \dots, x_u)$  be the number of elements in  $S$  for which the first  $u$  coordinates are  $(x_1, x_2, \dots, x_u)$ . The rank of  $\mathbf{x} \in S$  can be obtained by using

$$i_S(\mathbf{x}) = \sum_{j=1}^n x_j n_s(x_1, x_2, \dots, x_{j-1}, 0). \quad (1)$$

Cover’s enumeration scheme needs approximately  $kn$  weights for encoding and decoding  $(dklr)$  sequences because  $n_s(x_1, x_2, \dots, x_{j-1})$  only depends on the last  $k$  bits.

An alternative of the above enumeration scheme can be given by counting the number of elements of  $S$  that have a lexicographic index *higher* than  $\mathbf{x}$ , the *inverse rank* of  $\mathbf{x}$ . The inverse rank of  $\mathbf{x} \in S$  is given by

$$i_S^c(\mathbf{x}) = \sum_{j=1}^n \bar{x}_j n_s(x_1, x_2, \dots, x_{j-1}, 1) \quad (2)$$

where  $\bar{x}_j = 1 - x_j$ , the complement of  $x_j$ .

In the next section, taken from [16], we will apply the above theory to the enumeration of  $(dklr)$  sequences. We will first give an algorithm for enumerating  $(dkr)$ -constrained sequences for which the length of the leading zero-run is not constrained. From the results obtained, enumeration of  $(dklr)$  sequences follows easily.

##### A. Encoding and Decoding of $(dklr)$ Sequences

We introduce some notations that will facilitate the use of (2). Given a  $dkr$ -codeword  $\mathbf{x}$ , let  $\mathbf{x}_j^1 = (x_1, x_2, \dots, x_{j-1}, 1)$ . Denote by  $N^0(i)$  the number of  $(dkr)$ -constrained sequences of length  $i$  whose first element equals 1. We define the quantity  $a_j(\mathbf{x})$  as the length of the trailing zero-run of the subvector  $(x_1, \dots, x_{j-1})$  if it is not the all-zero sequence. Or

$$a_j(\mathbf{x}) = \begin{cases} \min\{(j - i - 1) : 1 \leq i < j, x_i = 1\}, & \text{if } j > 1 \text{ and } (x_1, \dots, x_{j-1}) \neq (0, \dots, 0) \\ d, & \text{otherwise.} \end{cases}$$

By invoking (2) to  $(dkr)$  sequences, we observe that for a given  $j$ ,  $1 \leq j \leq n$ ,  $n_s(\mathbf{x}_j^1)$  can take one of two different values, depending on  $\mathbf{x}$ . If  $a_j(\mathbf{x}) < d$  then  $n_s(\mathbf{x}_j^1) = 0$ , because in this case  $\mathbf{x}_j^1$  violates the prescribed  $d$  constraint. Alternatively, if  $a_j(\mathbf{x}) \geq d$  then  $n_s(\mathbf{x}_j^1) = N^0(n - j + 1)$ . The next proposition for the enumeration of  $(dkr)$  sequences is now immediate.

*Proposition 1:* The inverse rank of a  $(dkr)$  sequence  $\mathbf{x}$  of length  $n$  is

$$i_S^c(\mathbf{x}) = \sum_{j=1}^n \delta_j(\mathbf{x}) N^0(n - j + 1)$$

where

$$\delta_j(\mathbf{x}) = \begin{cases} 1, & \text{if } x_j = 0 \text{ and } a_j(\mathbf{x}) \geq d \\ 0, & \text{otherwise.} \end{cases}$$

In words, the inverse rank of  $\mathbf{x}$  can be obtained by summing the appropriate weight of each “zero” of  $\mathbf{x}$  that is either

contained in the leading zero-run or is preceded by at least  $d$  "zeros."

Clearly,  $(dkr)$ -sequences having more than  $l$  leading "zeros" have lower lexicographic indexes than  $(dklr)$  sequences. Therefore their inverse rank is higher. If we denote by  $\mathbf{x}_l$  the  $(dklr)$  codeword having the highest inverse rank in  $S$ , then the set  $\{0, \dots, i_S^c(\mathbf{x}_l)\}$  of inverse ranks corresponds to the set of all  $(dklr)$ -codewords of length  $n$ .

Proposition 1 provides the algorithm for decoding  $(dklr)$  sequences which can easily be translated into pseudocode. The pseudocode of the encoding operation, i.e., given the inverse lexicographic index  $I$  find the corresponding  $\mathbf{x}$ , is written below.

```

 $\hat{I} := I, a := d;$ 
for  $j = 1$  to  $n$  do
  if  $\hat{I} \geq N^0(n - j + 1)$  and  $a \geq d$ 
    then  $x_j := 0, \hat{I} := \hat{I} - N^0(n - j + 1)$ 
    else if  $a < d$ 
      then  $x_j := 0$ 
      else  $x_j := 1, a := -1;$ 
   $a := a + 1;$ 
end for

```

The computation of the weights  $N^0(n)$  is an exercise in combinatorics. An elegant computational method, which makes it possible to accurately approximate  $N^0(n)$  for large values of  $n$ , is based on generating functions (see Section V-A). Here we opt for a simple counting argument.

To that end, let  $U(n)$  be the number of  $(dk)$  sequences that start and end with a "1." Then the following recursions are immediate:

$$U(n) = \begin{cases} 0, & \text{if } n \leq 0 \\ 1, & \text{if } n = 1 \\ \sum_{i=d}^k U(n-i-1), & \text{if } n \geq 2. \end{cases} \quad (3)$$

The number of  $(dkr)$ -constrained sequences of length  $n, N^0(n)$ , is simply

$$N^0(n) = \sum_{i=0}^r U(n-i). \quad (4)$$

For  $n > d + k$  we may verify the following recursion:

$$N^0(n) = \sum_{i=d+1}^{k+1} N^0(n-i). \quad (5)$$

### B. Enumeration Using Floating-Point Arithmetic

With the enumeration algorithm discussed above we have, in principle, paved the way for a practical solution of the translation problem. However, a quick look at the hardware requirements will reveal that the method is not (yet) feasible. The hardware for implementing the enumeration algorithm comprises a (binary) adder, a subtracter, a comparator, and a lookup table of the precomputed set of weights  $\{N^0(i)\}$ ,  $1 \leq i \leq n$ . The binary fixed-point representation of a single weight requires  $Rn$  bits per weight, where  $R, 0 < R < 1$ , is a constant, which is approximately equal to the rate of the  $dk$

code. For the overall scheme, we need therefore a memory unit whose capacity grows proportional to  $Rn^2$ , which is prohibitive for the long codewords we have in mind. A second drawback of straightforward enumeration is that the additions in forming the weighted sum of the received binary word requires a double carry, which complicates the structure of a parallel adder. This renders the conversion of the adder from a parallel into a simple serial form practically impossible.

Next we will develop an enumeration method where the weights are specified in finite-precision floating-point notation. The floating-point notation is convenient for representing numbers that differ many orders of magnitude. In this notation, each weight is represented by  $s$  bits, and as a result, the hardware required for storage grows linearly with the codeword length  $n$ . The penalty attached to the finite-precision representation of the weights is that it will entail a (small) loss in code rate. A quantitative tradeoff between the precision of the number representation and concomitant code redundancy will be detailed in the next section.

We employ a two-part radix-2 representation

$$I = (m, e)$$

to express the weight

$$I = m \times 2^e$$

where  $I, m$ , and  $e$  are nonnegative integers. The two components  $m$  and  $e$  are usually called *mantissa* and *exponent* of the integer  $I$ , respectively. The translation of a weight into  $(m, e)$  is easily accomplished. It is assumed that the exponent of each weight is represented by  $e_p$  bits. With the following procedure we ensure that the mantissa  $m$  is represented by a specified number of bits, which we denote by  $p$ . Each weight is thus represented by  $s = e_p + p$  bits. There are various techniques such as rounding and truncation for translating fixed-point into floating-point representations. We will choose here a truncation operation; as it is, as we will show later, compatible with the basic enumeration algorithms.

Let  $I$  be one of the weights  $N^0(i)$ . It is well known that the nonnegative integer  $I, I < 2^v$ , can be uniquely represented by a binary  $v$ -tuple  $\mathbf{x} = (x_{v-1}, \dots, x_0)$ , where

$$I = \sum_{i=0}^{v-1} x_i 2^i.$$

The binary  $v$ -tuple  $\mathbf{x}$  is called the binary fixed-point representation of  $I$ . Let

$$u = \lfloor \log_2 I \rfloor$$

be the position of the leading "one" element of  $\mathbf{x}$ . Then the  $p$ -bit truncation of  $I$ , denoted by  $\lfloor I \rfloor_p$

$$\lfloor I \rfloor_p = \lfloor 2^{-(u+1-p)} I \rfloor 2^{u+1-p} \quad (6)$$

can be represented in binary floating-point representation whose mantissa requires at most  $p$  nonzero bits.

If the above finite-precision arithmetic is used in the enumeration algorithms we must modify the set of weights developed in the previous section. To that end, let  $\hat{N}^0(i)$

denote the number of  $(dkr)$  sequences of length  $i$  starting with a “1” that can be encoded with  $p$ -bit mantissa representation, then

$$\hat{N}^0(i) = \begin{cases} N^0(i), & i \leq k+d \\ \left\lfloor \sum_{j=d+1}^{k+1} \hat{N}^0(i-j) \right\rfloor_p, & 4 > k+d. \end{cases} \quad (7)$$

For clerical convenience, it is assumed that  $N^0(i) < 2^p$ ,  $1 < i \leq k+d$ , i.e., they can be represented by a mantissa of  $p$  bits; otherwise, more bookkeeping is required. The encoding and decoding algorithms developed in the previous section can be employed directly by using the “truncated” coefficients  $\hat{N}^0(i)$  in lieu of  $N^0(i)$ . The enumeration algorithm itself remains unchanged. The effect on the set of codewords will be that recursively the  $N^0(i) - \lfloor N^0(i) \rfloor_p$  highest ranking  $(dkr)$  words of length  $i$  are discarded from the set of all lexicographically ordered  $dkr$  sequences starting with a “1.”

## V. IMPLEMENTATION ASPECTS

It is of great engineering interest to know for what codeword length the rate of a code approaches channel capacity to within some small value, say 0.1%. To that end, we will, in this section, compute the penalties on code redundancy of both finite codeword length and finite precision of the weights. We start with an analysis of the effects of finite codeword length.

### A. How Long is Long Enough?

The number of  $(dklr)$  words,  $N_c(n)$ , of length  $n$ , available in one of the three block-code constructions described in Section III equals the coefficient  $a_n$  of the following generating function [17]:

$$\sum a_i x^i = \frac{q(x)}{p(x)} = \frac{x^{\beta+1}(1-x^{l+1})(1-x^{r+1})}{(1-x)(1-x-x^{d+1}+x^{k+2})}. \quad (8)$$

The parameters  $\beta, l$ , and  $r$  are stipulated by the construction recipe  $c_i$ ,  $i = 1, 2, 3$ , where  $c_i$  denotes Construction 1, 2, or 3, respectively. For large codeword length  $n$ , the number of codewords can be approximated by

$$N_c(n) \approx A_c \lambda^n \quad (9)$$

where  $\lambda$  is the largest real root of the characteristic equation

$$z^{k+2} - z^{k+1} - z^{k-d+1} + 1 = 0 \quad (10)$$

and the constant  $A_c$ , which is independent of  $n$ , equals

$$A_c = -\lambda \frac{q(1/\lambda)}{p'(1/\lambda)}. \quad (11)$$

Note that the channel capacity  $C(d, k)$  satisfies

$$C(d, k) = \log_2 \lambda.$$

Forsberg [17] found after evaluating many examples by computer experiments that the accuracy of (9) is surprisingly good. The rate of an implemented code is

$$R = \frac{1}{n} \lfloor \log_2 N_c(n) \rfloor. \quad (12)$$

TABLE I  
CONSTANT  $\log_2 A_c$  FOR SELECTED VALUES OF  $d$  AND  $k$

$d$	$k$	$C(d, k)$	$\log_2 A_{c1}$	$\log_2 A_{c2}$	$\log_2 A_{c3}$
1	3	0.551	-1.062	-0.357	
1	7	0.679	-0.771	-0.404	
2	7	0.517	-1.265	-0.602	-0.375
2	10	0.542	-1.039	-0.661	-0.332
3	10	0.446	-1.393	-0.807	-0.680

The terms  $C1, C2$ , and  $C3$  are short for Constructions 1, 2, and 3.

The difference between capacity and rate of the code is

$$\frac{1}{n}(-1 + \log_2 A_c) \leq R - C(d, k) \leq \frac{1}{n} \log_2 A_c. \quad (13)$$

In Table I we have collected results of computations.

From Table I we conclude that for the parameters of most practical interest the constant  $\log_2 A_c$  is in the range  $-0.5, \dots, -0.35$ . Thus a codeword length of approximately  $n = 256$  suffices to approach capacity to within 0.2–0.5%. As an example, we have computed  $C(d = 2, k = 15) - R$  as a function of the codeword length  $n$ . Results of computations are shown in Fig. 5. The diagram shows points for byte-oriented codes, i.e., codes whose source word length  $\log_2 \lfloor N_c(n) \rfloor$  is a multiple of eight. We restricted ourselves to byte-oriented codes to fit standard memory and code hardware.

### B. Effects of Finite-Precision Arithmetic

Using finite precision of the weights representation (truncation) will result in coding loss as available  $dklr$  words must be discarded. In this subsection we will study quantitatively the coding loss.

The number of  $(dkr)$  sequences of a length  $n$  that start with a “1,”  $N^0(n), n > k + d$ , equals (see (5))

$$N^0(n) = \sum_{i=d+1}^{k+1} N^0(n-i).$$

The number of  $(dkr)$  sequences grows exponentially with  $n$ , the growth factor being  $\lambda = 2^{C(d,k)}$ . Recall that for sufficiently large  $n$ , the number of  $(dkr)$  sequences of length  $n$ ,  $\hat{N}^0(n)$ , that can be encoded using a  $p$ -bit mantissa representation is

$$\hat{N}^0(n) = \left\lfloor \sum_{i=d+1}^{k+1} \hat{N}^0(n-i) \right\rfloor_p. \quad (14)$$

Conceptually, the computation of the growth factor  $\hat{\lambda}$  of  $\hat{N}^0(n)$  and the capacity  $\hat{C}(d, k) = \log_2 \hat{\lambda}$  becomes very simple if we notice that the sequence behavior of the  $p$ -bit mantissa of  $\hat{N}^0(n)$  versus  $n$  can be described in terms of an autonomous finite-state machine. The characteristic functions describing the sequence behavior of the finite-state machine are implied by the recursive equation (14). After a little thought the following Proposition will become clear.

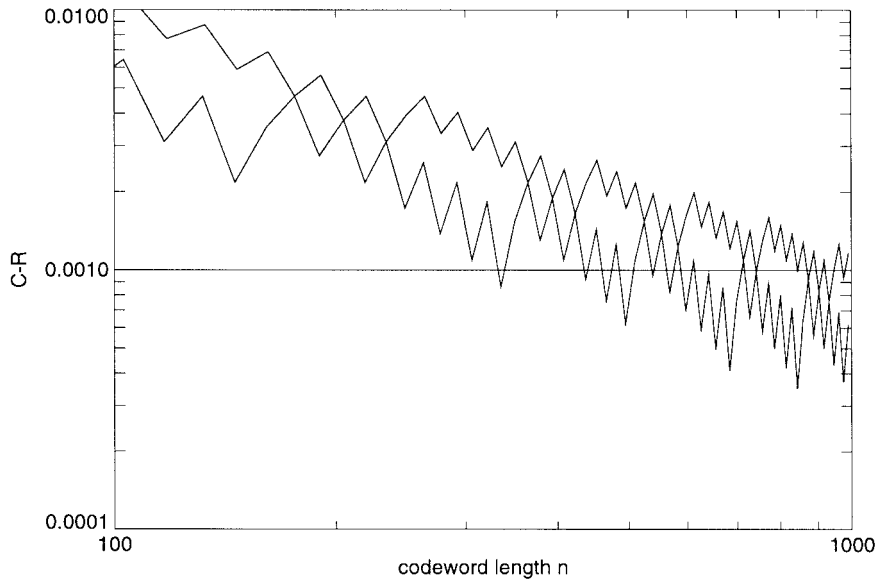


Fig. 5. Redundancy  $C - R$  versus codeword length of codes based on Construction 2 (upper curve) and Construction 3 (lower curve). The runlength parameters are  $d = 2$  and  $k = 15$ . The channel capacity is  $C(2, 15) = 0.5501$ . The diagram shows points for byte-oriented codes, i.e., codes whose source word length  $\log_2 \lfloor N_c(n) \rfloor$  is a multiple of eight. The “raggedness” of the curve is caused by the truncation to the nearest power of two.

TABLE II  
CAPACITY  $\hat{C}(d, k)$  FOR SELECTED VALUES OF  $d, k$ , AND  $p$

$d$	$k$	$C(d, k)$	$p$	$\hat{C}(d, k)$	$C(d, k) - \hat{C}(d, k)$
0	5	0.98811	7	62/63	0.00398
1	7	0.67929	8	40/59	0.00132
1	12	0.69299	9	9/13	0.00068
2	7	0.51737	9	61/118	0.00042
2	12	0.54711	7	6/11	0.00166
2	15	0.55011	11	11/20	0.00011
3	12	0.45555	7	5/11	0.00010

*Proposition 2:* The capacity  $\hat{C}(d, k)$  is rational.

*Proof:* From the theory of feedback registers [18] we know that the sequence of the mantissa of  $\hat{N}^0(n)$  will ultimately become (and remain) *periodic*. That is, there are integers  $h$  and  $f$  such that

$$\hat{N}^0(n)2^h = \hat{N}^0(n + f). \tag{15}$$

In other words, per-cycle period of length  $f$ , the number of sequences increases with a fixed factor, which is equal to a power of two  $2^h$ . From the above it is immediate that

$$\hat{C}(d, k) = \frac{h}{f}. \tag{16}$$

The theory of feedback registers [18] stipulates that the cycle period must be smaller than  $2^{p(k+2)}$ . As this number is huge in the range of parameters of practical interest, we are inclined to believe that Proposition 2 is not of great practical interest. However, results of a computer search, which are listed in Table II, reveal that relatively small cycle periods are surprisingly frequent.

Though the above method for computing the capacity  $\hat{C}(d, k)$  has the virtues of precision and efficiency, it does not provide a very useful relationship between  $p$  and the capacity loss  $C(d, k) - \hat{C}(d, k)$ . Here, following the heuristic analysis by Janssen, we try to obtain a relatively simple relationship without computer search.

Essentially, the loss in the number of sequences  $\hat{N}^0(n - i) - \lfloor \hat{N}^0(n - i) \rfloor_p$  at each step of the recursion is modeled as a random process. At each step we assume that, on average, a fixed portion  $\sigma_p$  of the number of sequences is lost. We linearize the nonlinear recursion (14) by

$$\hat{N}^0(n) = (1 - \sigma_p) \left( \sum_{i=d+1}^{k+1} \hat{N}^0(n - i) \right). \tag{17}$$

Note that it is a rather delicate matter whether we indeed can assume that the  $N^0(n)$  and  $\hat{N}^0(n)$  have the same asymptotic behavior. A more sophisticated approach would require to consider the  $\sum_{i=d+1}^{k+1} N^0(n - i)$  that have at most  $p + (k - d)$  nonzero bits to be generated by a finite-state machine, and to compute stationary distributions. This, however, is outside the scope of this paper. The limiting average relative truncation error  $\sigma_p$  is given by

$$\sigma_p = \lim_{N \rightarrow \infty} E \left[ \frac{\underline{x}(N) - \lfloor \underline{x}(N) \rfloor_p}{\underline{x}(N)} \right]$$

with  $\underline{x}(N)$  an integer random variable, assumed to be uniformly distributed on  $\{2^{N-1}, \dots, 2^N - 1\}$ . By elementary means it can be shown that

$$2^{-p} \ln 2 - 2^{-(2p+2)} \leq \sigma_p \leq 2^{-p} \ln 2.$$

The characteristic equation pertaining to the recursion (17) equals

$$\lambda^{k+2} - \lambda^{k+1} - (1 - \sigma_p)(\lambda^{k-d+1} - 1) = 0. \tag{18}$$

The difference between the largest root of (18) and (10) (and thus the difference between  $C(d, k)$  and  $\hat{C}(d, k)$ ) can be found as follows. Define

$$f(\lambda) = \frac{\lambda^{k+2} - \lambda^{k+1}}{\lambda^{k-d+1} - 1} = 1 - \sigma$$

and denote by  $\lambda(\sigma)$  the largest root of (18). Then, after linearizing  $f$  around  $\lambda(0)$ , we get

$$f(\lambda(\sigma)) = f(\lambda(0)) + (\lambda(\sigma) - \lambda(0))f'(\lambda(0)) = 1 - \sigma.$$

As  $f(\lambda(0)) = 1$  and  $\lambda(0) = \lambda$  we have

$$1 + (\lambda(\sigma_p) - \lambda)f'(\lambda(0)) = 1 - \sigma_p$$

so that

$$\hat{\lambda} = \lambda(\sigma_p) \approx \lambda - \frac{\sigma_p}{f'(\lambda)} = (1 - \sigma_p \mu)\lambda$$

where

$$\mu^{-1} = d + \frac{\lambda^{k+2} - (k+1-d)}{\lambda^{k+1-d} - 1}.$$

The capacity  $\hat{C}(d, k)$  can therefore be approximated by

$$\hat{C}(d, k) = \log_2 \hat{\lambda} \approx \log_2 \lambda - \frac{\sigma_p \mu}{\ln 2} \approx C(d, k) - \mu 2^{-p}$$

where we have replaced  $\sigma_p$  by  $2^{-p} \ln 2$ .

Practically, we have  $\mu \approx 0.25$  in the working range  $d = 1, \dots, 3, k \gg d$ , which means that the loss in capacity resulting from the truncation of the weights can be simply approximated by

$$C(d, k) - \hat{C}(d, k) \approx 2^{-(p+2)}. \quad (19)$$

A numerical comparison of the above result with the outcome of the method of computing the cycle time revealed that (19) can serve as a reliable rule of thumb.

The combined effects of the truncation and the finite code-word length on the achievable rate can be conveniently approximated by

$$C(d, k) - R \approx \frac{1}{2n} + 2^{-(p+2)}. \quad (20)$$

If both design parameters are chosen such that their relative effect on the rate loss is equal, i.e.,  $2n = 2^{p+2}$ , then we find the following fundamental relationship between hardware (storage) requirements and efficiency of a (long)  $dk$  code:

$$C(d, k) - R \approx \frac{1}{n}. \quad (21)$$

Preferably a ROM is used as a lookup table. Then the number of ROM output bits  $p$  is one less than the number of input bits  $\lceil \log_2 n \rceil$ . We found that, to some extent, the above results do not depend on the specified runlength parameters  $d$  and  $k$ . As a rule of thumb, we may safely conclude that we can design in the main case of practical interest, where  $d < 3$ , any  $(dk)$ -constrained code whose rate is only 0.1% below capacity with the aid of a ROM lookup table having 10 input and 9 output bits. In other words, with a hardware of 1-kByte ROM one can construct a block code whose rate practically equals channel capacity. In the above analysis it is tacitly assumed that storage is only needed for the mantissa of the weights. The exponent of the weights must also be stored, but as this additional storage is proportional to  $\log n$ , it is neglected.

It should be appreciated that often a significant saving in storage hardware can simply be realized by noting that the mantissa of the weights ultimately becomes periodic. By a

judicious choice of  $k$  and  $p$  we may attempt to reduce the hardware requirements by minimizing the cycle period. Table II reveals that short cycle periods (and thus small storage hardware) are possible for values of  $d$  and  $k$  of practical interest. How to systematically approach the minimization of the cycle period is not clear yet.

In the above, it is assumed that the set of weights is precalculated and stored in ROM. Alternatively, it is, of course, feasible to compute the set of weights “on-the-fly” as the computation of a weight requires at most  $k-d+1$  floating-point additions. Unfortunately, this is only of potential benefit during the decoding process. Note that during encoding the algorithm calls the largest weights first, while the computation of the weights starts with the smallest weights.

## VI. WORKED EXAMPLES

The effectiveness of the new technique will be illustrated by a few worked examples.

### A. Alternatives to Standard Codes

In this subsection, we will compare codes constructed by the new coding technique with traditional sliding-block codes: the rate  $2/3$ ,  $(1, 7)$  code and the rate  $1/2$ ,  $(2, 7)$  code [19], which have been used in magnetic disk drives. The maximum runlength constraint  $k$  is imposed to restrict the maximum time between two consecutive transitions in the recorded signal. The  $k = 7$  constraint can be relaxed to improve the code rate. Practical values are  $k = 12$  for the  $d = 1$  code and  $k = 15$  for the  $d = 2$  as a further increase would lead to at most 0.2% saving in rate. Using Construction 2, it is straightforward to design a rate  $256/371$ ,  $(1, 12)$  code,  $p = 9$ . The efficiency of this code,  $R/C(1, 12) = 0.996$ . The saving in code rate offers a serviceable 3.5% increase in storage capacity compared to its traditional predecessor. Assume this scheme is used in conjunction with a byte-oriented RS code. Prior to forwarding the constrained sequence to the ECC it is compressed. It can easily be verified that  $q = 11$  is the largest word length for which the number of  $d = 1$ ,  $k = 12$  sequences of a length  $q$  is less or equal 256. Thus the compression scheme translates the 371-bit  $dk$  sequence into  $\lceil 371/11 \rceil = 34$  bytes. The expansion of the ECC input sequence, with respect to the conventional “Fig. 1” configuration, is thus  $34/32 = 1.06$ .

Using Construction 3, it is possible to design the alternative rate  $256/466$ ,  $(2, 15)$  code,  $p = 11$ . The efficiency of this code,  $R/C(2, 15) = 0.9986$ . The rate of the new code is almost 10% larger than that of the traditional code. A possible lossless compression scheme has an input word length  $q = 13$ . Then the 466-bit input sequence is translated into  $\lceil 466/13 \rceil = 36$  bytes, which results in an expansion factor of  $36/32 = 1.125$ . Alternatively, we may opt for a more complex compression scheme, which translates  $q = 28$  bits into 2 bytes. This results in a smaller expansion factor, 1.06, but the increased error propagation (a single channel bit error may result in two decoded byte errors) puts an extra load on the ECC. The trading of the various parameters is a subtle matter requiring more study.

TABLE III  
REDUNDANCY  $1 - C(0, k)$  VERSUS  $k$

$k$	$1 - C(0, k)$
1	0.30576
2	0.12085
3	0.05322
4	0.02477
5	0.01189
6	0.00581
7	0.00287
8	0.00142

TABLE IV  
MAXIMUM CODEWORD LENGTH  $\hat{n}$  FOR WHICH A  
RATE  $1 - 1/\hat{n}$ ,  $(0, k)$  CODE CAN BE CONSTRUCTED

$k$	$\hat{n}$
4	31
5	67
6	148
7	310
8	649

The hardware requirements of both codes are, considering the potential of current electronics technology, quite moderate. Perusal of Table II shows that both codes have parameters that lead to small cycle periods, which, in turn, can be exploited to reduce the size of the lookup table. For example, for the rate 256/371, (1, 12) code we find a cycle time  $f = 13$ . Including the ‘‘preamble’’ weights we require room for 52 weights instead of 370 weights in case of the straightforward implementation. In the next section we will focus on coding  $(k)$ -constrained sequences.

B. Application to  $(k)$ -Constrained Sequences

The capacity  $C(0, k)$  equals  $\log_2(\lambda)$ , where  $\lambda$  is the largest root of [2]

$$x^{k+2} - 2x^{k+1} + 1 = 0. \tag{22}$$

For sufficiently large  $k$ , we derive

$$\lambda \approx 2 \left( 1 - \frac{1}{2^{k+2}} \right)$$

so that

$$C(0, k) \approx 1 - \frac{1}{\ln 2} 2^{-k-2}, \quad k \gg 1. \tag{23}$$

Table III lists the redundancy  $1 - C(0, k)$  versus the runlength parameter  $k$ .

In Table IV we have listed the maximum codeword length  $\hat{n}$ , and redundancy  $1 - 1/\hat{n}$  as a function of the maximum runlength parameter  $k$  when the coefficients are not truncated.

In order to maximize the number of words, we have set  $r = \lceil k/2 \rceil$  and  $l = k - r$  (see [17]). We see that the redundancy required is very close to the bound listed in Table III.

The effect of weight truncation can be seen in Table V, where we have listed the maximum codeword length  $\hat{n}$  and redundancy  $1 - 1/\hat{n}$  as a function of the runlength parameter  $k$ . The redundancy is only slightly larger than in the case of full representation of the weights (see Table II).

TABLE V  
MAXIMUM CODEWORD LENGTH  $\hat{n}$  FOR WHICH A RATE  
 $1 - 1/\hat{n}(0, k)$  CODE,  $p = k + 2$ , CAN BE CONSTRUCTED

$k$	$\hat{n}$
4	26
5	54
6	112
7	232
8	474

The mantissa of the weights  $\hat{N}^0(i)$  can be represented by at most  $p > k$  bits if we use

$$\hat{N}^0(i) = \begin{cases} 0, & i \leq 0 \\ 2^{i-1}, & i = 1, \dots, r+1 \\ \sum_{j=i-1-k}^{i-1} \hat{N}^0(j), & i = r+2, \dots, p+1 \\ 2^{i-p-1} \left[ 2^{p-i+1} \sum_{j=i-1-k}^{i-1} \hat{N}^0(j) \right], & i > p+1. \end{cases} \tag{24}$$

In the special case,  $p = k + 2$ , we find that mantissa and exponent of the floating-point representation of the weights are simple functions

$$\hat{N}^0(i) = \begin{cases} 2^{i-1}, & i = 1, \dots, r+1 \\ (2^{r+1} - 1)2^{i-r-2}, & i = r+2, \dots, k+2 \\ (a_0 - i)2^{i-k-3}, & k+2 < i \leq i_1 \end{cases} \tag{25}$$

where

$$a_0 = (2^{r+1} - 1)2^{k-r+1} + k + 2 \\ i_1 = a_0 - 2^{k+1}.$$

The expressions for  $\hat{N}^0(i), i > i_1$ , are too involved and therefore omitted.

VII. CONCLUSIONS

We have presented a new configuration of codes that makes it possible to use long constrained codewords without the risk of massive error propagation. To illustrate the effectiveness of the new configuration we have studied long  $(dk)$ -constrained codes. We have introduced a scheme of enumerative coding of  $dk$  sequences using floating-point arithmetic. This scheme offers, for long codewords, a significant advantage in storage requirements. A simple relationship between coding efficiency versus encoding or decoding hardware (storage) has been derived. For most common  $(dk)$  constraints, we found that a code rate of less than 0.5% below channel capacity can be obtained by using hardware consisting of a ROM lookup table of 1 kbyte. For selected values of  $d$  and  $k$ , we can exploit the structure (periodicity) of the weights, which lead to significant savings in storage requirements. As an illustrative numerical example we presented a rate 256/466,  $(d = 2, k = 15)$  block code, which provides a serviceable 10% increase in rate with respect to its traditional rate  $1/2$ ,  $(2, 7)$  counterpart.

## ACKNOWLEDGMENT

The author wishes to thank A. J. Janssen, Philips Research Labs, for providing the analysis on the effects of truncating coefficients described in Section V-B. He also wishes to thank Dr. A. R. Calderbank, Editor-in-Chief, and an anonymous reviewer, who provided valuable comments on an earlier version of the manuscript.

## REFERENCES

- [1] S. B. Wicker and V. K. Bhargava, Eds., *Reed-Solomon Codes and Their Applications*. New York: IEEE Press, 1994.
- [2] K. A. S. Immink, *Coding Techniques for Digital Recorders*. Englewood Cliffs, NJ: Prentice-Hall Int. (UK) Ltd., 1991.
- [3] M. Blaum, "Combining ECC with modulation: Performance comparisons," *IEEE Trans. Inform. Theory*, vol. 37, pp. 945–949, May 1991.
- [4] T. M. Cover, "Enumerative source coding," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 73–77, Jan. 1973.
- [5] W. G. Bliss, "Circuitry for performing error correction calculations on baseband encoded data to eliminate error propagation," *IBM Tech. Discl. Bul.*, vol. 23, pp. 4633–4634, 1981.
- [6] M. Mansuripur, "Enumerative modulation coding with arbitrary constraints and post-modulation error correction coding and data storage systems," *Proc. SPIE*, vol. 1499, pp. 72–86, 1991.
- [7] J. L. Fan and A. R. Calderbank, "A modified concatenated coding scheme with applications to magnetic recording," submitted to *IEEE Trans. Inform. Theory*, 1996.
- [8] D. T. Tang and L. R. Bahl, "Block codes for a class of constrained noiseless channels," *Inform. Contr.*, vol. 17, pp. 436–461, 1970.
- [9] G. F. M. Beenker and K. A. S. Immink, "A generalized method for encoding and decoding runlength-limited binary sequences," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 751–754, Sept. 1983.
- [10] J. Gu and T. Fuja, "A new approach to constructing optimal block codes for runlength-limited channels," *IEEE Trans. Inform. Theory*, vol. 40, pp. 774–785, May 1994.
- [11] K. A. S. Immink, "Constructions of almost block-decodable runlength-limited codes," *IEEE Trans. Inform. Theory*, vol. 41, pp. 284–287, Jan. 1995.
- [12] K. A. S. Abdel-Ghaffar and J. H. Weber, "Bounds and constructions for runlength-limited error-control block codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 789–800, May 1991.
- [13] K. W. Cattermole, *Principles of Pulse Code Modulation*. London, U.K.: Iliffe Books, 1969.
- [14] E. Labin and P. R. Asgrain, "Electric pulse communication system," U.K. Patent 713 614, 1951.
- [15] W. H. Kautz, "Fibonacci codes for synchronization control," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 284–292, 1965.
- [16] L. Patrovics and K. A. S. Immink, "Encoding of  $dklr$ -sequences using one weight set," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1553–1554, Sept. 1996.
- [17] K. Forsberg and I. F. Blake, "The enumeration of  $(d, k)$  sequences," in *Proc. 26th Allerton Conf. on Communications, Control, and Computing* (Montecello, IL, Sept. 1988), pp. 471–472.
- [18] S. W. Golomb, *Shift Register Sequences*. San Francisco, CA: Holden-Day, 1967.
- [19] T. D. Howell, "Statistical properties of selected recording codes," *IBM J. Res. Devel.*, vol. 33, no. 1, pp. 60–73, Jan. 1989.