

BLOCK-DECODABLE RUNLENGTH-LIMITED CODES VIA LOOK-AHEAD TECHNIQUE

by Kees A. SCHOUHAMER IMMINK

Philips Research Laboratories, P.O. Box 80000, 5600 JA Eindhoven, The Netherlands

Abstract

The terms (d,k) -constrained sequence and runlength-limited (RLL) sequence are usually used as synonyms, and traditionally the design of encoders that generate RLL sequences is almost always conducted by designing encoders that generate (d,k) -constrained sequences followed by a precoder. It is generally believed that this design procedure does not entail a loss of performance in terms of coder complexity and error propagation. In this paper, however, we will show that it is surprisingly profitable in terms of error propagation to design RLL encoders directly, i.e. without the intermediate step of a (d,k) -constrained sequence.

Keywords: digital recording, error propagation, modulation codes, state combining algorithm

1. Introduction

Since the early 1970s, coding methods based on (d,k) -constrained sequences have been widely used in such high-capacity storage systems as magnetic and optical disks or tapes. Properties and applications of (d,k) -constrained sequences, or runlength-limited (RLL) sequences as they are often called, are surveyed in ref. 1. The number of sequential-like symbols in a (binary) sequence is known as *runlength*. A (d,k) RLL sequence is a sequence of binary symbols characterized by two parameters, $(d+1)$ and $(k+1)$ which stipulate the minimum and maximum runlength, respectively, that may occur in the sequence. Closely related to RLL sequences are (d,k) sequences. A binary sequence is said to be (d,k) constrained if the number of "zeros" between any pair of consecutive "ones" is at least d and at most k , $k > d$. A (d,k) -constrained sequence can be converted into a (d,k) RLL sequence by a simple coding step which is known as *precoding*. The ones in the (d,k) -constrained sequence indicate the positions of a transition $1 \rightarrow 0$ or $0 \rightarrow 1$ of the corresponding

runlength-limited sequence. It can readily be verified that the minimum and maximum runlength of the RLL sequence derived from a (d,k) sequence via precoding is $d+1$ and $k+1$ symbols, respectively.

Codes are used to translate source data into the constrained sequence. Commonly, the source data is partitioned into words of length m , and under the coding rules these m -tuples are translated into n -tuples called codewords. Popular (d,k) codes incorporated in disk file systems are the $(2,7)$ and the $(1,7)$ codes of rate $m/n = 1/2$ and rate $2/3$, respectively². The codes are designed using the bounded delay method^{3,4,5} or the ACH sliding-block code algorithm⁶. The principal feature of a (d,k) (or other finite-type constraints) code constructed with the sliding-block code algorithm is that coded sequences can be decoded by examining a limited number of consecutive symbols without relying on external state information. As an immediate consequence, these codes have a limited amount of error propagation. For example, a single bit error in a received sequence encoded by the $(2,7)$ sliding-block code propagates at most over four decoded bits, while a bit error in the $(1,7)$ code propagates at most over five decoded bits. Blaum⁷ showed that the error propagation of sliding-block codes presents a problem as it entails an extra load to the error correction circuitry usually used in conjunction with the (d,k) code.

An alternative to the above sliding-block coding scheme was proposed by Tang and Balaf⁸, and Franaszek⁹. There, the authors use codes compiled from codewords of fixed length which can be decoded without the knowledge of preceding or succeeding codewords. Codes with this property, i.e. codes that can be decoded by observing single codewords (the encoding operation is allowed to be state dependent), will be called *block(-decodable) codes*. Evidently, block-decodable codes offer an advantageous solution relative to sliding-block codes since they make it easier to preserve a particular mapping between the source and the code symbols, and, obviously, error propagation is localized to one decoded m -block. Block-decodable codes are highly suitable in conjunction with Reed-Solomon error control codes. In the preferred embodiment of the coding system, the codewords have a 1-1 correspondence with the elements of the finite field $GF(2^n)$, thus enabling the construction of, for instance, a Reed-Solomon code directly over the (d,k) -constrained codewords. A notable drawback of state-of-the-art block-decodable code constructions, however, is the fact that at code rates $R = m/n$ approaching the Shannon capacity of the (d,k) -constrained channel, the implementations can be fairly complex, involving long codewords. For example, the minimum codeword lengths allowing a rate $R = 2/3$, $(1,7)$ block-decodable code and a rate $R = 1/2$, $(2,7)$ block-decodable code are 33 and 34, respectively. If the

system requirements make it possible to relax the k constraint, we can design an $R = 8/16$, $(2,10)$ block code. This block code is particularly attractive as many commercially available Reed-Solomon codes operate in $GF(2^8)$. A modification of this code, involving suppression of the low-frequency components, is employed in the Compact Disc¹⁰. A rate $m/n = 8/12$, $(1,k)$ block-decodable code has not been published in the literature. The question arises: can we construct such a block-decodable code (or other codes) for the given (or other) parameters? Our approach is based on the observation that such a code must be constructed on RLL sequences rather than (d,k) sequences. In the literature, the terms (d,k) sequence and RLL sequence are usually used as synonyms, and the design of encoders that generate RLL sequences is almost always conducted by designing encoders that generate (d,k) sequences followed by a precoder. Sequences that are assumed to be recorded with such an intermediate precoding step are said to be given in *non-return-to-zero-inverse* (NRZI) notation, whereas sequences transmitted without such a precoding step are referred to as *non-return-to-zero* (NRZ). Coding techniques using the NRZI format are generally accepted in digital optical and magnetic recording practice. The NRZI format is convenient in magnetic recording since differentiation occurs as part of the physical process in the inductive heads. Each magnetic transition on the medium generates a corresponding pulse, a "peak", in the read-back signal. The original binary NRZI signal can be restored in quite a natural fashion by observing the position of the peaks in the read-back signal. The peaks coincide with the ones of the stored sequence written in NRZI notation. The use of the NRZI format in non-differentiating, i.e. full-response, channels, such as the Compact Disc or magnetic recorders with magneto-resistive read heads, is less obvious. The reason for adopting NRZI notation in such full-response channels is merely convenience, since the polarity of the received sequence is irrelevant, and therefore not a part of an adopted standard. It is generally believed that the choice of the NRZI format does not entail a loss of performance in terms of coder complexity and error propagation. In this paper, however, we will show that it is surprisingly profitable in terms of error propagation to design RLL encoders directly, i.e. without the intermediate step of a (d,k) -constrained sequence. The new RLL codes to be discussed are block decodable, while at the same time they are simpler to implement than (d,k) block-decodable codes currently being used.

We start, in the next section, with a description of the basic idea of the new coding technique. In Sec. 3, we will present a general and systematic design tool for constructing constrained block encoders employing the single-word look-ahead technique. As a result, numerous new block-decodable codes with

TABLE I
Codebook of a $d = 1$ RLL block code.

Source	Channel representation
0	0000
1	0001/0110/1000
2	0011
3	0111/1001/1110
4	1100
5	1111

relatively short block lengths will be presented in Sec. 4. Specifically, we will present a new rate $8/12$, $(1,9)$ RLL block-decodable code and a new rate $8/9$, $(3,4)$ PRML block-decodable code (see later for a definition), which are both well adapted to byte-oriented storage systems.

2. Description of the basic idea

In this section, we will describe the basic idea of the new coding technique. In order to simplify our exposition, it is convenient to confine ourselves for a while to sequences with a minimum runlength only. A generalization to sequences with both a minimum and a maximum runlength constraint is postponed to a later section. The following simple example will demonstrate the potential usefulness of the new coding technique, and in fact it motivated us to pursue the work described in the sequel to this paper.

Consider the $(d = 1, \infty)$ RLL code described in Table I. The left hand column shows the source symbols, represented by an integer between 0 and 5, while the right hand column shows the possible channel representations of the corresponding source word. Note that the RLL code is presented in NRZ notation, i.e. sequences generated by this code should therefore not be followed by a precoding operation. Clearly, the codeword length is four and the code is size six. As we may observe, four of the source words, namely 0, 2, 4 and 5, are uniquely represented by a codeword. It is easily verified that these four codewords can be freely concatenated without offending the prescribed minimum runlength constraint. The remaining two source words are each represented by three alternative codewords whose choice depends on the last codeword transmitted and the next codeword to come. For this specific case, it is easily verified that by looking ahead and looking back at most one

codeword in time, it is always possible to select (at least) one of the three alternative representations that fulfils the stated minimum runlength requirement. The above coding rules can be cast into the model of a standard finite-state encoder. The encoder graph has eight states, and the output and next-state functions can be represented by the following 8×8 matrix:

0/0000	1/0000	2/0000	3/0000	4/0000	5/0000
0/0110		2/0110		1/0001	3/0001
0/0011		2/0011		1/0011	3/0011
0/0111		2/0111		1/0111	3/0111
0/1000	1/1000	2/1000	3/1000	4/1000	5/1000
0/1110		2/1110		1/1001	3/1001
0/1100	1/1100	2/1100	3/1100	4/1100	5/1100
0/1111		2/1111		1/1111	3/1111

The matrix elements are expressed in the form x/y , where x is the encoder input and y is the corresponding encoder output. For reasons of space, we follow the convention that, if a transition is allowed from state i to state j , the ij matrix element equals the label x/y pertaining to the edge $i \rightarrow j$; if, on the other hand, such a transition is not permitted, the matrix element is empty. The look-ahead dependence of the codewords was accounted for by tagging edge labels taken from the look-ahead code (Table I), where the codewords are delayed by one block interval. It should be appreciated that the standard finite-state encoder given above is not a preferred embodiment of the coding rules. It is usually advantageous to translate the code table plus the look-back and look-ahead dependences directly into logical gates using a CAD package rather than minimizing the number of encoder states. A casual glance at the encoder table is sufficient to convince the reader that the table has the virtue of a unique state-independent inverse, which means that observation of a single codeword is sufficient information for the retrieval of the corresponding source word. We conclude, therefore, that the code presented in Table I is a $(1, \infty)$ RLL block-decodable code. In contrast, a conventional (d,k) -constrained block-decodable code with the same parameters constructed with Franaszek's method⁹ has size five, and we draw the fascinating conclusion that the above RLL block-decodable code has a slightly larger size.

The code presented in the preceding example can be generalized for arbitrary values of the codeword length $n > 3$. The size of the code, denoted by $N(n)$, is given by

TABLE II
Code size $N(n)$ of $(1, \infty)$ RLL block codes of codeword length n along with the code size, $F(n-1)$, of conventional (d, k) codes of the same codeword length.

n	$N(n)$	$F(n-1)$
4	6	5
5	10	8
6	17	13
7	27	21
8	44	34
9	72	55
10	116	89
11	188	144
12	305	233

$$\begin{aligned}
 N(n) &= 0, \quad n < 0, \\
 N(0) &= 1, \\
 N(n) &= N(n-3) + F(n-1), \quad N > 0,
 \end{aligned} \tag{1}$$

where the Fibonacci numbers $F(n)$ are given by

$$\begin{aligned}
 F(n) &= 0, \quad n < 0, \\
 F(0) &= 1, \quad F(1) = 2, \\
 F(n) &= F(n-1) + F(n-2), \quad n > 1.
 \end{aligned} \tag{2}$$

Despite the elegance of the recursive expression, we could only prove (1) after a straightforward, but tedious, process of enumerating and, therefore, the proof is omitted. In Table II, we have tabulated $N(n)$ as a function of the codeword length n . The size of the (d, k) -constrained block code of the same codeword length, $F(n-1)$, is included for comparison purposes.

Table II reveals that the smallest rate $2/3, (d-1)$ RLL block-decodable code has a codeword length $n = 6$. For the same rate, the conventional (d, k) block code requires a codeword length of at least 18 (see for example ref. 1). We also note that $N(9) = 72$, which suggests the feasibility of the construction of a rate $6/9, (1, k < \infty)$ RLL block-decodable code and, of even more practical relevance, that $N(12) = 305$, which suggests the feasibility of the construction of

a rate $8/12, (1, k < \infty)$ RLL block code with the attractive source word length $m = 8$. Both codes are, as can be seen in Table II, not possible with conventional (d, k) -constrained block codes of the same length.

The hard part, to be discussed in the subsequent sections, is systematically to design the above, and other, codes incorporating a maximum runlength constraint.

3. Code design with single-word look-ahead

The following section serves mainly to introduce a glossary of concepts and notation that will be employed later.

3.1. General

Consider the construction of an encoder which converts source (or input) sequences $\{b_i\}$ to constrained (or output) channel sequences $\{x_i\}$ and a decoder which reconstitutes $\{x_i\}$ into the original $\{b_i\}$. The source sequence is partitioned into m -tuples $\{b_i\}$, called *source words*, and the output sequence is partitioned into n -tuples $\{x_i\}$, called *codewords*. The encoder can be modelled as a finite-state automaton defined by two characteristic functions:

$$\begin{aligned}
 x_i &= h(b_1, \dots, b_{i+a}, s_i), \\
 s_{i+1} &= g(b_1, \dots, b_{i+a}, s_i),
 \end{aligned} \tag{3}$$

where s_i is a member of a finite set called the *state set* of the encoder. The parameter $a, a > 0$, is called the *look-ahead span* of the encoder. A code constructed by the ACH algorithm can be decoded by a sliding-block decoder. A sliding-block decoder is given by a function

$$b_i = f(x_{i-q}, \dots, x_{i+r}), \tag{4}$$

where q is the *memory* and r the *anticipation* of the decoder. Essentially, the decoder comprises an n -stage register of length $(q+r+1)$ that stores the $(q+r+1)$ retrieved n -bit codewords and a (Boolean) function $f(\cdot)$ that translates the contents of the register into the retrieved m -bit source word b_i . Since the constants q and r are finite, an error in the retrieved sequence $\{x_i\}$ can propagate in the sequence $\{b_i\}$ only for a finite distance, at most the decoder window length $(r+q+1)$. The relationship between the window length and the various code parameters is still an open problem. In this paper, we will be interested primarily in codes whose decoding function is characterized by $q = r = 0$. In other words, these codes have the virtue that error propagation is limited to exactly one decoded m -bit block.

There are a large variety of design tools for constructing constrained channel codes. The most notable ones are the ACH algorithm⁶), the bounded delay (BD) method⁵), and the look-ahead (LA) coding technique¹¹). The three design techniques produce codes of somewhat different form (a recent paper by Franaszek¹²) may be consulted as an entry into the literature), but their starting point is the same, namely a finite-state transition diagram representing the channel constraints. A finite-state transition diagram is a labelled directed graph with a finite number of states and edges with labels. The symbol V is used to denote the set of states, and the symbol E_{ij} to denote the set of labels attached to the edges that have the starting state i and the ending state j . The labels are length- n codewords. We define the $|V| \times |V|$ transition matrix T with elements $t_{ij} = |E_{ij}|$, where $|x|$ denotes the cardinality of the set x . The design procedures proceed by computing an *approximate eigenvector* v . An approximate eigenvector $v = (v_1, \dots, v_{|V|})^T$, whose elements v_i are non-negative integers (not all 0s), satisfies the approximate eigenvector inequality

$$Tv \geq \alpha v, \quad (5)$$

where (in)equality means componentwise (in)equality. The approximate eigenvector plays a key role in both the ACH algorithm and the new algorithm. A simple algorithm, introduced by Franaszek⁴), can be used to compute approximate eigenvectors. The component v_i is called the *weight* of state i , and the scalar α is a positive integer. For practical codes, we have $\alpha = 2^m$. Let λ denote the largest positive eigenvalue of T . Then, provided $\alpha \leq \lambda$, the existence of an approximate eigenvector satisfying (5) is guaranteed by the tenets of the Perron-Frobenius theory of non-negative matrices¹³).

Since look-ahead is an essential element of the example codes described in the previous section, we have focused our research on an algorithm for designing constrained codes that employ one-block look-ahead during encoding.

3.2. A new design algorithm

The new algorithm is succinctly described as follows.

(1) Set up, as outlined in the previous section, a (directed) graph describing the given runlength constraints. The starting graph has a transition matrix with 0/1 elements, and it is *memoryless*, i.e. each label attached to an edge of such a graph uniquely identifies the initial state of its corresponding edge. A transition matrix with 0/1 elements can always be found by going to the edge graph (see ref. 6, Theorem 5.1) or by letting the admitted length- n codewords serve as states. Other characteristics of the starting graph will become apparent in the sequel.

(2) Given the state-transition matrix T and a convenient $\alpha \geq 2^m$ compute an approximate eigenvector v . States having zero weight are removed by crossing out corresponding rows and columns of T .

The two previous steps of the algorithm are exactly the same as used in the ACH algorithm. The ACH algorithm proceeds with a process of state splittings and state mergings. After a state splitting operation, the old state is replaced by two (or more) new states. Each new state has the same incoming edges as the original state, and the outgoing edges from the original state are partitioned among the new states. The aim of the state splitting procedure is to transform the graph into a new graph with transition matrix \hat{T} with the property that, compared with T , either the maximum weight of states of \hat{T} or the number of states with maximum weight has been reduced. The splitting process is performed a finite number of rounds with the role of the new T played by \hat{T} until an approximate vector \hat{v} is reached having all its components equal to zero or one (not all 0s). States having zero weight are removed by crossing out rows and columns of \hat{T} corresponding to the weightless states. According to ref. 6, it is always possible to find a sequence of state splittings that converges to the "all-ones" eigenvector. Then we have found a graph each of whose states has row sums at least α , i.e. each state has at least α outgoing edges. The construction procedure is concluded by tagging the outgoing edges with the α source words; if necessary excess edges are deleted so that every row sum equals α . Redundant states can be removed by a process called *state merging*.

(3) In contrast with the ACH algorithm, the new algorithm proceeds with a process called *state combining*. The aim of the state combining procedure, the *main strategy* is to obtain a new graph with transition matrix \hat{T} and approximate eigenvector \hat{v} with the property that compared with T either the minimum non-zero weight of states of \hat{T} or the number of states with minimum non-zero weight has been increased, and where the maximum weight, $\max\{\hat{v}_i\}$, should not be larger than the maximum weight, $\max\{v_i\}$, of the states of the old graph. This process is applied recursively, with the role of the new T played by \hat{T} until we reach the point where we have a vector \hat{v} whose components equal either zero or $\max\{v_i\}$. If we divide both sides of the approximate eigenvector inequality (5) by $\max\{v_i\}$, we see that this is equivalent, with an approximate eigenvector vector having all its components equal to zero or one. The advantage of this approach is that state combinations can be found, as will be outlined shortly, that do not increase the anticipation (or the memory) of the decoder, whereas, as a general rule, each round of state splittings in the ACH algorithm increases the decoder anticipation by one.

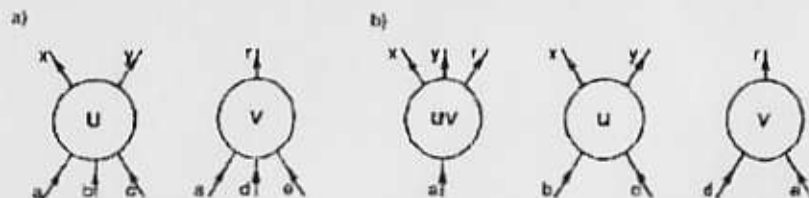


Fig. 1. Local picture of a state combination a) before the combination, and b) after the combination of states u and v .

We will first, in the next section, provide a description of the state combination procedure; thereafter, we will proceed with the design algorithm.

3.2.1. State combination procedure

The (u, v) combination of states u and v , called the *constituent states* is obtained by the following procedure.

- Define a new state uv and connect it to the successor states of u and v .
- States $w \in V$ having both u and v as a successor and whose edges connecting $w \rightarrow u$ and $w \rightarrow v$ have the same label A are connected to uv with the label A .
- Edges emanating from states $w \in v$ having both u and V as a successor as well as having the same label are eliminated.

The state combination is vacuous if the set of edges entering the combined state uv is empty, and it is therefore not considered. After invoking the above procedure, the new graph has one more state than the old one. Figure 1 illustrates the foregoing state combination procedure.

A (u, v) state combination is said to be *strategic* if it is consistent with the main strategy. Obviously, a (u, v) state combination is strategic if the weight of the combined state does not exceed the maximum weight $\max\{v_i\}$ and if it is the sum of the non-zero weights of the two constituent states u and v and if the weights of the remaining states $i \in V \setminus \{u, v\}$ are unchanged. This requirement can be captured by the following condition, called the *vertical condition*:

$$\sum_{j \in V \setminus \{u, v\}} v_j t_{ij} + (v_u + v_v) |E_u \cap E_v| \geq w_i, \quad \forall i \in V. \quad (6)$$

It is easily verified that the above condition ensures that a) the weight of the newly created state equals the sum of the weights of its constituent states, b) the constituent states can be dropped from the graph since they are weightless, and c) the weights of the remaining states are left unchanged. Loosely speaking, the vertical condition guarantees that two light-weight states can be

changed into a state whose weight is the sum of the weights of the "old" states without altering the weights of the remaining states. In the remainder of the paper, unless otherwise indicated, it will be assumed that a (u, v) state combination is followed by a removal of the states u and v . Obviously, after invoking the above procedure, the number of states of the newly constructed graph is one less than the original graph. If, after a (u, v) combination, the new graph preserves the Shannon capacity of the original graph, we say that the (u, v) combination is *lossless*. A state combination should not be confused with a state merger. Both operations reduce the number of states by one. An important distinction, however, is that the weight of a merged state equals the weight of its constituent states while, in our context, the weight of a combined state has a weight equal to the sum of the weights of its constituent states.

(4) If, after a sequence of state combinations, a graph is found with an all-ones approximate eigenvector (weightless states are removed), we can proceed to the final part of the algorithm, namely the assignment of source words to the available edges at each state. If, on the other hand, such an approximate eigenvector is not found, we start from the top again by attempting alternative state combinations. The algorithm does not guarantee success in finding the desired situation. The above algorithm is best explained with a simple example. The purpose of this example is a) to illustrate the fact that the new algorithm is not restricted to the design of RLL codes, and b) that, in general, the codes obtained are not block-decodable codes.

Example 1: Consider a conventional $(0,1)$ -constrained code with block lengths $m = 2$ and $n = 3$. The given parameters can be cast into a 5-state graph represented by the matrix

$$\begin{matrix} 1 & & & 010 & 010 & 010 \\ 2 & 011 & 011 & 011 & 011 & 011 \\ 3 & 101 & 101 & 101 & 101 & 101 \\ 4 & & & 110 & 110 & 110 \\ 5 & 111 & 111 & 111 & 111 & 111 \end{matrix}$$

The left-hand column gives the state numbers. Recall that we follow the convention that, if a transition from state i to state j is admissible, the i, j matrix element equals the label pertaining to the edge $i \rightarrow j$; if, on the other hand, such a transition is not permitted, the matrix element is empty. Note that the labelled graph is not a familiar unifilar, or deterministic, graph, where for each state $v \in V$ the outgoing edges of v are assumed to be distinctly labelled. It is

easily verified that the given graph is of the Moore type, where each state can be associated with a distinct state label. In order to incorporate the one-block delay, we have tagged the state labels to the outgoing edges instead of the conventional tagging to the incoming edges. The resultant graph is *backwards deterministic*, i.e. for each state $u \in V$ the edges entering u are distinctly labelled. It is easily verified that

$$r^T = (1, 2, 2, 1, 2)$$

is an approximate eigenvector for $\alpha = 2^n = 4$. After the (1,4) state combination, we obtain the 6-state graph

$$\begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 1+4 \end{array} \left[\begin{array}{cccc} & 010 & 010 & 010 \\ 011 & 011 & & 011 & 011 \\ 101 & 101 & & 101 & 101 \\ 110 & 110 & & 110 & \\ 111 & 111 & & 111 & 111 \\ 010 & 010 & 010 & & \\ 110 & 110 & 110 & & \end{array} \right]$$

where the combined state is denoted by "1+4". In order to illustrate the procedure, we did not remove the constituent states. It is easily verified that, in the new graph, state 1 is degenerate as it has no incoming edges, and state 4 is weightless. After discarding states 1 and 4, we obtain the 4-state graph

$$\left[\begin{array}{cccc} 011 & 011 & 011 & 011 \\ 101 & 101 & 101 & 101 \\ 111 & 111 & 111 & 111 \\ 010 & 010 & & \\ 110 & 110 & & \end{array} \right]$$

In the above graph, each state has exactly row sum four, so that we can immediately construct an encoder graph by assigning, for each state v , $\alpha = 4$ distinct tags (source words) to the $\alpha = 4$ outgoing edges from v . The code found is equivalent to a variable-length code with a maximum word length of six bits; decoding can be accomplished with a sliding-block decoder of window length two (codewords). \square

As seen in the above example, codes generated by the above algorithm are, in general, not block-decodable codes. The reason is the ambiguity caused by the

fact that there are parallel edges connecting states. Since our main objective is the design of block-decodable codes, we have to cure this difficulty by extending the algorithm with an extra condition. Based on the above observation, the encoder graph of a block-decodable code must satisfy the condition that at most one edge is allowed between states, i.e. the entries of the state-transition matrix must be either one or zero. If a (u,v) state combination might lead to the situation of two parallel edges connecting states, implying $t_{uj} = t_{vj} = 1$, $j \in V$, then only one edge may be kept. After eliminating excess edges, however, the new graph with state-transition matrix \hat{T} must satisfy $\hat{T}\hat{v} \geq \alpha\hat{v}$, where the new approximate eigenvector \hat{v} should be in line with the main strategy. Specifically, the weight of the combined state must be the sum of the weights of the two constituent states. This condition can be formulated as follows. A (u,v) combination of states u and v is strategic if both the vertical and the horizontal condition hold:

$$\sum_{j \in V} v_j \phi(t_{uj}, t_{vj}) \geq \alpha(v_u + v_v), \quad (7)$$

where $\phi(x,y) = 0$, if $x = y = 0$ otherwise $\phi(x,y) = 1$. If $t_{uj} = t_{vj} = 1$, an arbitrary choice is made to delete the excess edge. The algorithm provides freedom in the choice of which of the two edges will be dropped.

If, after application of a finite number of state combinations satisfying both the horizontal and the vertical conditions, followed by dropping the weightless states, a final graph is found with an all-ones approximate eigenvector, we may proceed to the assignment part of the algorithm. As it is assumed that we start with a memoryless graph whose transition matrix consists of zeros and ones, it can easily be verified that a (final) graph obtained after a sequence of state combinations preserves these properties. Note that this is not the case if combinations are performed without discarding the constituent states. A mapping of the set of source words to the set of states, the *source-to-state assignment*, concludes the construction of the encoder. There are, in general, more encoder states than source words, so that it is not *a priori* certain whether a source-to-state assignment can be found that admits the desired unique inverse. If an invertible source-to-state assignment can indeed be found, then, since the encoder graph is memoryless, there is a unique relationship between any state and its outgoing labelled edges, and we have discovered a code with the virtue that it can be decoded by observing single blocks.

In summary, the state combination algorithm applied to a memoryless graph will produce a block-decodable code if a) we can find an all-ones approximate eigenvector, and b) if an invertible source-to-state assignment can be found. Apparently, we are confronted with a massive computational

problem: both the state combination process and the source-to-state assignment are essentially problems that must be handled by brute force. The assignment problem is related to the "graph K-colorability problem", which is known to be NP complete¹⁴). However, it will be shown shortly that for many cases of practical interest block-decodable codes can be found fairly easily.

The next example reveals how the RLL block-decodable code presented in Sec. 2 can be found in the systematic manner described above.

Example 2: Consider a $(1, \infty)$ RLL code composed of codewords of length four. A 10×10 binary transition matrix describing the constraints is

0000	0000	0000	0000	0000	0000	0000	0000
					0001	0001	0001
0011	0011	0011			0011	0011	0011
0110	0110	0110	0110	0110			
0111	0111	0111			0111	0111	0111
1000	1000	1000	1000	1000	1000	1000	1000
					1001	1001	1001
1100	1100	1100	1100	1100	1100	1100	1100
1110	1110	1110	1110	1110			
1111	1111	1111			1111	1111	1111

Let $\alpha = 6$. Then, we find the approximate eigenvector

$$r^T = (2, 1, 2, 1, 2, 2, 1, 2, 1, 2).$$

Pairs of states with unity weight are combined. There are four states of unity weight, and thus three feasible permutations of state pairs. However, after invoking the horizontal and vertical conditions, we conclude that only one such permutation is strategic, namely combinations (2,4) and (7,9). After performing these combinations followed by some bookkeeping, we find an 8-state graph represented by

0000	0000	0000	0000	0000	0000
0110	0110	0110	0110	0001	0001
0011		0011		0011	0011
0111		0111		0111	0111
1000	1000	1000	1000	1000	1000
1110	1110	1110	1110	1001	1001
1100	1100	1100	1100	1100	1100
1111		1111		1111	1111

Note that each state has a row sum of at least $\alpha = 6$. A source-to-codewords assignment, discarding the excess edges leaving states 2 and 6, with a unique inverse can be found. The result is the encoder described in Sec. 2.

The next example shows a code design where a unique inverse could not be found.

Example 3: Consider a rate $2/3$, $(1, \infty)$ RLL code. The starting point of the code construction is the binary transition matrix

000	000	000	000	000
			001	001
011	011		011	011
100	100	100	100	100
110	110	110		
111	111		111	111

Let $\alpha = 4$; then the approximate eigenvector is

$$r^T = (2, 1, 2, 2, 1, 2).$$

After the (2,5) state combination, we obtain

000	000	000	000
110		110	001
011	011		011
100	100	100	100
111	111		111

Each state has a row sum of $\alpha = 4$, but a source-to-codewords assignment with a unique inverse cannot be found.

4. Results

The algorithm outlined in the previous section has been implemented and successfully applied to find new block-decodable codes. In order to reduce the computer load, we have restricted the code parameters in such a way that the entries of the approximate eigenvector v , are at most two; the general case is postponed to a later study.

4.1. RLL codes

Numerous new RLL block-decodable codes were found for runlength parameters and word lengths m and n for which no conventional (d,k) block-decodable codes are known. A brief survey of new RLL block-decodable codes of practical interest is listed in Table III. For comparison purposes we have tabulated in parentheses the shortest possible codeword length of conventional block-decodable (d,k) codes of the same rate and runlength parameters. The rate 8/12, (1,9) and the rate 8/16, (2,8) RLL block-decodable codes are of specific interest as they are well adapted to byte-formatted storage systems.

As remarked, the algorithm may require much computation and the examination of candidate state combinations to find a block-decodable code for an arbitrary choice of parameters. However, we discovered that in many instances only a few permutations of state combinations have to be tried, and that the state combination process quite rapidly converges to the "all-ones" vector. The computer program, guided by the saying "all good things in life

TABLE III
Survey of new RLL block-decodable codes.

d	k	m	n	
0	1	4	6	(18)
0	2	5	6	(12)
1	∞	4	6	(18)
1	10	6	9	(21)
1	9	8	12	(21)
2	∞	3	6	(14)
2	8	8	16	(22)
3	10	6	15	(20)

are free", starts by performing all possible lossless state combinations. In general, only a few alternatives remain to be tried. For example, the rate 8/12, (1,9) code presented in Table III, requires initially 460 states. There are 196 states of unity weight of which 164 can be combined without loss. Only 2 min of main-frame computer time were needed to find strategic combinations for the remaining 32 unity-weight states as well as to find and check an invertible assignment. The required number of encoder states, 358, may look prohibitively large, but this is merely the result of the standard finite-state machine description chosen in the algorithm. Given the finite-state encoder, we can easily construct a code book as depicted in Table I, and this code book can in turn, using the look-back and look-ahead features, be implemented in a straightforward manner. The required hardware is well within the reach of modern LSI. Admittedly, the code is much more complex than the rate 2/3, (1,7) (d,k) code, but any coding scheme is merely a part of a larger system and its cost must be in proportion to its importance within that system. These system considerations are beyond the scope of this paper.

4.2. PRML codes

The algorithm was also applied to code constraints for partial-response channels employing maximum-likelihood detection (PRML)^{15,16}. In prior art coding techniques, the channel constraints are described using NRZI notation, i.e. the constraints take the form of limitations on the maximum number of sequential zeros. A PRML-coded sequence, when NRZ recording is used, is a $(0,k)$ -constrained RLL sequence satisfying the characteristics that the maximum runlength of both the substrings of even and odd numbered symbols which interleave to make up the code string is $k_i + 1$. A small value of k is desirable for accurate timing and gain control, and a small value of k_i reduces the size of the path memory in the maximum-likelihood detection. There is no need to reduce inter-symbol interference by imposing a d -constraint. After application of the new design algorithm, a rate 8/9, $(k = 3, k_i = 4)$ PRML block-decodable code in NRZ notation was discovered, whereas a code (in NRZI notation) with the same parameters found by the ACH algorithm has the disadvantage relative to the new code that it requires a sliding-block decoder with a window length of two codewords (see Table III¹⁷).

5. Conclusions

We have developed a new algorithm for designing constrained codes having the virtue that error propagation is limited to exactly one decoded m -bit block. Various new look-ahead RLL block-decodable codes with relatively short

block lengths have been found which are simpler to implement than conventional (d,k) block codes currently being used. We have shown that it is surprisingly profitable in terms of error propagation to design RLL codes directly, i.e. without the intermediate step of a (d,k) -constrained code followed by a precoding operation. Specifically, we have presented a new rate 8/12, (1,9) RLL block-decodable code and a new rate 8/9, (3,4) PRML block-decodable code, which are both well adapted to byte-oriented storage systems.

Acknowledgement

The author is indebted to colleagues, Henk Hollmann in particular, for many fruitful discussions.

REFERENCES

- ¹⁾ K.A.S. Immink, Coding Techniques for Digital Recorders, Prentice-Hall, Englewood Cliffs, NJ, 1991.
- ²⁾ T.D. Howell, IBM J. Res. Dev., 33 (1), 60-73 (1989).
- ³⁾ P.A. Franaszek, IBM J. Res. Dev., 23 75-81 (1979).
- ⁴⁾ P.A. Franaszek, IBM J. Res. Dev., 24 628-641 (1980).
- ⁵⁾ P.A. Franaszek, IBM J. Res. Dev., 26 506-514 (1982).
- ⁶⁾ R.L. Adler, D. Coppersmith and M. Hassner, IEEE Trans. Inform. Theory, IT-29 5-22 (1983).
- ⁷⁾ M. Blaum, IEEE Trans. Inform. Theory IT-37 (3) 945-949 (1991).
- ⁸⁾ D.T. Tang and L.R. Bahl, Inform. Control, 17 436-461 (1970).
- ⁹⁾ P.A. Franaszek, IBM J. Res. Dev., 14 376-383 (1970).
- ¹⁰⁾ J.P.J. Heemskerk and K.A.S. Immink, Philips Tech. Rev., 40 (6), 157-164 (1982).
- ¹¹⁾ A. Lempel and M. Cohn, IEEE Trans. Inform. Theory, IT-28 933-937 (1982).
- ¹²⁾ P.A. Franaszek, IBM J. Res. Dev., 33 (6), 602-608 (1989).
- ¹³⁾ R.S. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1962.
- ¹⁴⁾ M.R. Garey and D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979.
- ¹⁵⁾ J.S. Eggenberger and A.M. Patel, Method and apparatus for implementing optimum PRML codes, US Patent 4707681, November 1987.
- ¹⁶⁾ B.H. Marcus, A.M. Patel and P.H. Siegel, Method and apparatus for implementing a PRML code, US Patent 4786890, November 1988.
- ¹⁷⁾ B.H. Marcus, P.H. Siegel and J.K. Wolf, Finite-state modulation codes for data storage, IEEE J. Selected Areas Commun., 10 (1992).

Authors



Kees A. Schouhamer Immink was born in Rotterdam, the Netherlands, in 1946. He received a B.S. degree from the Rotterdam Polytechnic in 1967, and M.S. and Ph.D. degrees from Eindhoven University of Technology in 1974 and 1985, respectively, all in electrical engineering. He joined Philips Research Laboratories, Eindhoven, in 1968, where his work involved the signal processing side of optical recording systems, later becoming responsible for the design and development of channel coding techniques for the Compact Disc, Compact Disc Video, experimental erasable optical audio discs, R-DAT and DCC recorders. In 1986, he was appointed senior scientist of the Philips Research magnetic recording group where his current research focuses on digital magnetic audio and video recorders for consumer applications. He is a Fellow of the AES, IEE (London) and IEEE.