Statistical properties of selected recording codes

by Thomas D. Howell

Most recording systems encode their data using binary run-length-limited (RLL) codes. Statistics such as the density of 1s, the probabilities of specific code strings or run lengths, and the power spectrum are useful in analyzing the performance of RLL codes in these applications. These statistics are easy to compute for ideal run-length-limited codes, those whose only constraints are the run-length limits, but ideal RLL codes are not usable in practice because their code rates are irrational. Implemented RLL codes achieve rational rates by not using all code sequences which satisfy the run-length constraints, and their statistics are different from those of the ideal RLL codes. Little attention has been paid to the computation of statistics for these practical codes. In this paper a method is presented for computing statistics of implemented codes. The key step is to develop an exact description of the code sequences which are used. A consequence of the code having rational rate is that all the code-string and run-length probabilities are rational. The method is illustrated by applying it to three codes of practical importance: MFM, (2, 7), and (1, 7).

1. Introduction

Most digital magnetic and optical recording systems use binary run-length-limited (RLL) codes [1–4]. These codes are described by pairs of run-length limits, (d, k). The parameter d is the minimum number of symbols **0** between consecutive symbols **1** in the encoded data, and k is the maximum. The playback system is designed to sense the presence or absence of **1**s in the encoded data. The parameters (d, k) are chosen to place adjacent **1**s far enough apart to avoid problems with intersymbol interference, yet close enough together to ensure accurate clock recovery. The *capacity* or *entropy* of a set of run-length constraints is an upper bound on the rate of any code satisfying those constraints. Equivalently, it is the maximum number N of unconstrained bits which can be encoded by each constrained symbol. Maxentropic (or ideal) codes have rates equal to the capacity of the constraints. For large N, they use all allowable code strings of length N about equally often. In practice, ideal RLL codes cannot be implemented. They can only be approximated.

It is often necessary to compute statistical properties of the coded data. Properties such as the frequencies of occurrence of particular strings of code symbols or of the various allowable run lengths of **0**s are useful in analyzing the performance of codes in some applications. For example, the frequency of occurrence of the symbol **1** might be required in analyzing the performance of a clock recovery system. All these properties are easy to compute under the assumption that the code is ideal [5–7]. Some tools for computing properties of ideal codes are summarized in Section 2.

Practical codes, the ones actually used in digital recording systems, are not ideal RLL codes. The capacities of all

interesting ideal RLL codes are irrational. The only exceptions are $d = k$ with a capacity of zero and $d = 0$, $k = \infty$ with a capacity of one bit per symbol (see [8]). Practical codes have rational rates; they encode m user bits into n constrained symbols. The integers m and n are typically quite small. The rate $r = m/n$ must be less than the capacity C of the original constraints (usually it is only slightly below, within a few percent). The ratio r/C is the *efficiency* of the code. The higher the efficiency, the better the code approximates the ideal RLL code.

Computing statistical properties for most practical codes is more difficult than it is for ideal codes. Two approaches are often taken. One is to compute the analogous quantities for the ideal code. One then argues that since the rate m/n is close to the capacity C, the statistical properties should also be close (see [6, 9, 10]). The second approach is simulation. One encodes a long string of random data and observes the statistical properties of the resulting constrained string. Each of these methods can be useful, but rigorous error bounds are rarely given.

Practical RLL codes typically do not use all allowable constrained sequences. Some sequences which satisfy the (d, k) constraints never appear in encoded data. In many cases of interest, though, the remaining sequences are used in the ideal way. That is, every string of a given, sufficiently large length is used with the same frequency if it is used at all. Such codes are really ideal codes whose constraints consist of run-length constraints and other incidental constraints. This class of codes includes the codes used in most computer disk drives, magnetic and optical. Understanding the incidental constraints is the key to computing the statistics of practical RLL codes. In many cases the incidental constraints are interesting for other reasons as well. They describe the set of strings which satisfy the run-length constraints but are not used in the actual code. Such strings are sometimes used as markers. They are readable because they have proper run lengths, but they never occur in valid data, and the decoder can flag them as code violations. In Section 3 the incidental constraints for some important codes are given, and this information is used to compute statistical properties of the codes.

2. Ideal run-length-limited codes

Ideal run-length constraints (d, k) may be described by a labeled directed graph, $\mathbf{G}(d, k)$. The graph has nodes labeled $0, 1, \cdots, k$. Each node i except $i = k$ has an outgoing edge to node $i + 1$ with label **0**. Each node i with $i \geq d$ has an outgoing edge to node 0 with label **1**. See **Figure 1**. The sequence of edge labels encountered while following any path in $\mathbf{G}(d, k)$ forms a (d, k)-constrained binary sequence. Conversely, any (d, k)-constrained binary sequence corresponds to a path in the graph. The label of each node gives the number of edges labeled **0** traversed since the last edge labeled **1**.

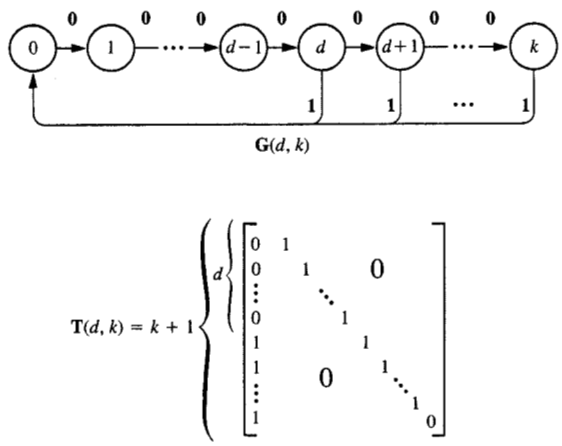


Figure 1

The graph $\mathbf{G}(d, k)$ describing the (d, k) run-length-limited binary sequences, and its adjacency matrix $\mathbf{T}(d, k)$.

The adjacency matrix of $\mathbf{G}(d, k)$ will be called $\mathbf{T}(d, k)$. Its entry in row i and column j gives the number of edges from node i to node j for $0 \leq i, j \leq k$. Information theory tells us how to compute statistical properties of the ideal sequences described by such a graph or adjacency matrix. The following results are well known. Some of the proofs can be found in [5], [7], or [11].

1. The capacity of the constraints described by adjacency matrix $\mathbf{T}$ is given by $C = \log_2 \lambda$ bits per constrained symbol, where λ is the largest real eigenvalue of $\mathbf{T}$. When $\mathbf{T}$ is $\mathbf{T}(d, k)$, it is easy to show that the characteristic polynomial is $x^{k+1} - x^{k-d} - x^{k-d-1} - \cdots - x - 1$. Its largest real root is λ.
2. Probabilities can be attached to the edges in the constraint graph in such a way that capacity is achieved when the probability of a given constrained string is given by the product of the probabilities assigned to the edges of the corresponding path in the graph. The probability assigned to an edge from node i to node j is given by $p_{ij} = (v_j/\lambda v_i)$, where v is the right eigenvector for $\mathbf{T}$ with eigenvalue λ: $\mathbf{T}v = \lambda v$. For $\mathbf{T} = \mathbf{T}(d, k)$, the right eigenvector is given by*

$$v^{\dagger} = [1\ \lambda\ \lambda^2 \cdots \lambda^d(\lambda^{d+1} - 1)(\lambda^{d+2} - \lambda - 1) \cdots (\lambda^k - \lambda^{k-d-1} - \cdots - \lambda - 1)].$$

The matrix $\mathbf{Q}$ whose (i, j) element is $q_{ij} = p_{ij}t_{ij}$ gives the

*The superscript † denotes a transpose.

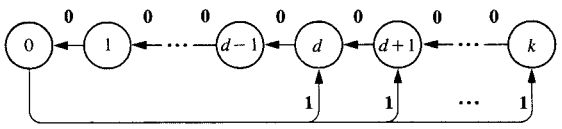


Figure 2

The reverse of $\mathbf{G}(d, k)$. The probabilities for the edges leaving node **0** are the same as the run-length probabilities for the ideal (d, k) code.

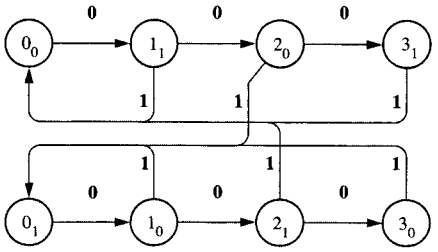


Figure 3

The graph $\mathbf{G}(1, 3)$ converted to be cyclic of index 2.

probability that node j follows node i. It is stochastic:

$$\sum_j q_{ij} = 1.$$

3. The probability s_i associated with node i of the constraint graph, **G**, is given by $w_i v_i$, where w and v are left and right eigenvectors for **T** with eigenvalue λ, and $\Sigma s_i = 1$. The vector s is also a left eigenvector for **Q** satisfying $s^\dagger = s^\dagger \mathbf{Q}$. For $\mathbf{T} = \mathbf{T}(d, k)$, the left eigenvector, w, is given by

$$w^\dagger = [\lambda^k \lambda^{k-1} \cdots 1],$$

and s is given by $s = \hat{s}/\Sigma \hat{s}_i$, where $\hat{s}_i = \lambda^k$ for $0 \le i \le d$ and

$$\hat{s}_i = \lambda^k - \sum_{j=1}^{i-d} \lambda^{k-d-j}$$

for $d + 1 \le i \le k$.

Given the node and edge probabilities, it is easy to calculate the probabilities of specific sequences of channel symbols. As an example, we calculate the probabilities of all three-symbol strings in the ideal (1, 3) code. We have $p(\mathbf{011}) = p(\mathbf{110}) = p(\mathbf{111}) = 0$ from the constraint $d = 1$, and

$$\begin{aligned}
p(\mathbf{000}) &= s_0 p_{01} p_{12} p_{23} &&= s_3 &&= 0.07879,\\
p(\mathbf{001}) &= s_0 p_{01} p_{12} p_{20} + s_1 p_{12} p_{23} p_{30} &&= s_2 &&= 0.19425,\\
p(\mathbf{010}) &= s_0 &&= s_0 &&= 0.36348,\\
p(\mathbf{100}) &= s_0 p_{01} p_{12} &&= s_2 &&= 0.19425,\\
p(\mathbf{101}) &= s_0 p_{01} p_{10} &&= s_0 p_{10} &&= 0.16923.
\end{aligned}$$

Note that $p(\mathbf{001}) = p(\mathbf{100})$.

Ideal RLL codes are symmetrical with respect to reversal. Reversing the directions of all edges in $\mathbf{G}(d, k)$ and reversing their labels yields a graph whose adjacency matrix is the transpose of $\mathbf{T}(d, k)$. The reversed graph describes the same set of constrained strings as the original. The node probabilities in the reversed graph are the same as in the original graph, but the edge probabilities now depend on the left eigenvector w. The probability assigned to an edge from node i to node j in the reversed graph is given by $\tilde{p}_{ij} = (w_j/w_i\lambda)$, where w is the left eigenvector for **T** with eigenvalue λ. It is shown in Appendix A that the probabilities of any code string and its reversal are equal in a reversal-symmetric, maxentropic code. The reverse of $\mathbf{G}(d, k)$ is shown in **Figure 2**.

The reversed graph provides a very simple way to compute the probabilities of the possible run lengths in an ideal RLL code. The probability of run length g, $d \le g \le k$, is just the edge probability $\tilde{p}_{0g}$ from node 0 to node g in the reversed graph. Using the formulas $\tilde{p}_{ij} = (w_j/w_i)\lambda^{-1}$ and $w_i = \lambda^{k-i}$ from 2) and 3) above, we find $\tilde{p}_{0g} = \lambda^{-(g+1)}$. The probability of a phrase of length $g + 1$ consisting of a **1** followed by g **0**s starting and ending at node 0 is just $\lambda^{-(g+1)}$. The probability of any string of independently chosen phrases whose total length is N is λ^{-N}. This illustrates that the given probabilities lead to a code which is indeed maxentropic.

3. Practical run-length-limited codes

Practical run-length-limited codes operate at code rates of the form m/n, where m and n are small integers. Since ideal RLL codes have irrational rates, practical RLL codes are not ideal. In most cases of interest they differ from ideal RLL codes by the addition of a few constraints. We will call these *incidental constraints*. Once the incidental constraints are known, the methods of the previous section can be used to compute node and edge probabilities for the graph describing the new code. The trick is to define incidental constraints such that the capacity of the new code is m/n. When the encoding rules for the code are known, one can derive the incidental constraints. Otherwise, a lemma of

Ashley and Siegel [8] is helpful in determining what incidental constraints might lead to capacity m/n. It states that any irreducible graph **G** representing constraints whose capacity is equal to m/n, where m and n are relatively prime integers, has the property that the number of edges in every cycle is a multiple of n. A graph is irreducible if there is a path from every node to every other node. A cycle is a path which begins and ends at the same node. We will say that an irreducible graph for which the greatest common divisor of the cycle lengths is n is *cyclic of index n*.

One might hope to construct codes by looking for incidental constraints with rational capacity m/n. The lemma of Ashley and Siegel gives a necessary condition which limits the search to constraints whose graphs are cyclic of index n. Finding constraints with rational capacity is not easy even in this very restricted set of graphs. All nontrivial examples of incidental constraints with rational capacities have come from analyzing codes constructed by other methods. In this context, existing code construction methods can be viewed as ways of searching systematically for incidental constraints with rational capacities.

Our strategy for computing the statistical properties of practical RLL codes consists of three steps:

1. Convert **G**(d, k) to a graph which describes the same constrained sequences and is cyclic of index n.
2. Modify this new graph according to the incidental constraints.
3. Compute the node and edge probabilities for the maxentropic code described by the modified graph.

This process is illustrated by applying it to three codes: MFM, (1, 7), and (2, 7).

• *The MFM code*

The MFM (modified frequency modulation) code is a rate 1/2 (1, 3) RLL code. It has several other names, including "delay modulation" and "Miller code." It can be analyzed more simply than is done here, and its properties are known [3, 12]. It is included to illustrate the techniques which we extend to the more complicated (1, 7) and (2, 7) codes.

We begin the construction of MFM with **G**(1, 3). Since we are aiming for a rate 1/2 code, we need to modify **G**(1, 3) to be cyclic of index 2. This is done by interconnecting two copies of **G**(1, 3) as shown in **Figure 3**. The adjacency matrix of the resulting graph is the Krönecker product of a permutation matrix corresponding to a 2-cycle with **T**(1, 3),

$$C_2 \otimes \mathbf{T}(1, 3) = \begin{bmatrix} 0 & T(1, 3) \\ T(1, 3) & 0 \end{bmatrix}.$$

This graph describes the same set of constrained strings as **T**(1, 3). It is cyclic of index 2 because the subscripts alternate on the labels of the consecutive nodes reached on any path. A path can circle back to its starting node only after an even number of edges; this property is inherited by any subgraph, so it is possible that the capacity of the constraints represented by some subgraph is exactly 1/2. In fact, the graph produced by deleting node 3_0 and the edges incident to it has this property. This modification corresponds to the incidental constraint that runs of three zeros beginning at odd-numbered indices are prohibited. The graph resulting from this modification is shown in **Figure 4**.

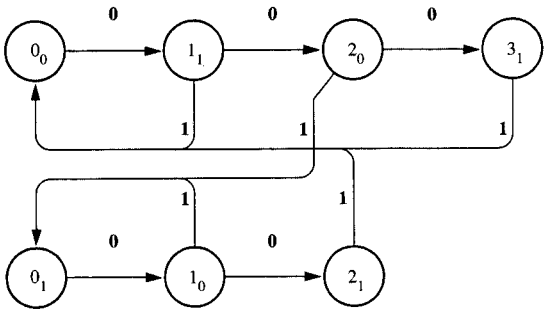


Figure 4

The modified graph with node 3_0 deleted.

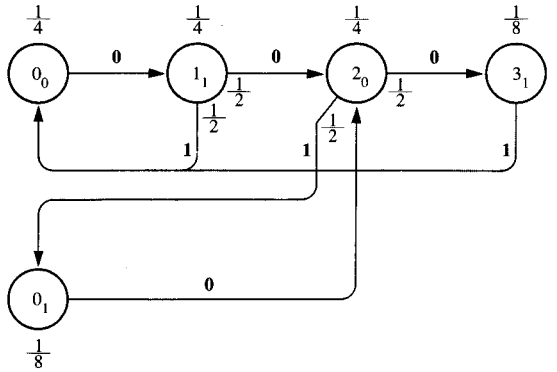


Figure 5

The MFM constraint graph.

The modified graph can be simplified. Nodes 3_1 and 2_1 have the same outgoing edges, so they can be merged into a single node. The same process can then be applied to nodes 2_0 and 1_0. The resulting graph is our basic description of the MFM constraints. The node and edge probabilities can be computed by the methods of Section 2. **Figure 5** shows the

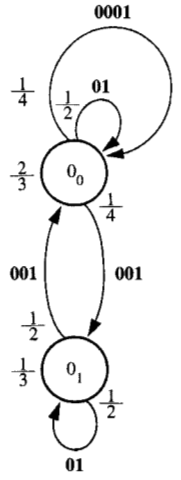


Figure 6

The MFM run-length graph.

graph with its associated probabilities. Note that all node and edge probabilities for MFM are rational numbers, while many of the corresponding probabilities for the ideal (1, 3) code are irrational. The fact that node and edge probabilities are rational for codes with rational rates is proven in Appendix B.

The MFM constraint graph contains all the information about the statistics of the MFM code. From it we can compute probabilities of arbitrary strings and probabilities of the different run lengths. For example, we give the probabilities of all three-symbol strings for comparison with those given in Section 2 for the ideal (1, 3) code:

$$p(\mathbf{000}) = s_{1_1} p_{1_1 2_0} p_{2_0 3_1} = 0.0625,$$

$$p(\mathbf{001}) = p(\mathbf{100}) = 0.1875,$$

$$p(\mathbf{010}) = s_{0_0} + s_{0_1} = 0.375,$$

$$p(\mathbf{100}) = s_{1_1} p_{1_1 2_0} + s_{0_1} p_{0_1 2_0} p_{2_0 3_1} = 0.1875,$$

$$p(\mathbf{101}) = s_{1_1} p_{1_1 0_0} + s_{0_1} p_{0_1 2_0} p_{2_0 0_1} = 0.1875.$$

The fact that $p(\mathbf{001}) = p(\mathbf{100})$ follows from the fact that the MFM constraints are symmetrical with respect to reversal.

The run-length probabilities are most easily found by constructing another graph, the run-length graph. The MFM run-length graph is the projection of the MFM constraint graph onto its nodes with labels 0_i. It has only two nodes, 0_0 and 0_1. It has one edge for each path in the MFM constraint graph that begins at one of these nodes and ends at one of these nodes without passing through either of these nodes along the way. The label of such an edge is the concatenation of the labels on the edges along the path in the MFM constraint graph. It always has the form of a run of zeros followed by a **1**. The MFM run-length graph describes exactly the same set of constrained strings as does the MFM constraint graph. It highlights the structure of these strings as sequences of phrases, where each phrase is a run of zeros terminated by a **1**. The node and edge probabilities for the MFM run-length graph can be computed directly by the methods of [5]. Alternatively, they can be determined quite easily from the corresponding quantities in the MFM constraint graph. The node probabilities for the run-length graph are the corresponding node probabilities from the constraint graph, renormalized so that they sum to one. The edge probabilities in the run-length graph are the products of the edge probabilities along the corresponding paths in the constraint graph. The MFM run-length graph is shown in **Figure 6**.

The run-length probabilities from each node can be read directly from the run-length graph. The overall probability of a given run length is the sum of the probabilities of that run length from each node, weighted by the node probabilities. The run-length probabilities for MFM are

$$p(\mathbf{01}) = \frac{1}{2}\frac{2}{3} + \frac{1}{2}\frac{1}{3} = \frac{1}{2},$$

$$p(\mathbf{001}) = \frac{1}{4}\frac{2}{3} + \frac{1}{2}\frac{1}{3} = \frac{1}{3},$$

$$p(\mathbf{0001}) = \frac{1}{4}\frac{2}{3} = \frac{1}{6}.$$

The power spectrum of a code is the Fourier transform of its autocorrelation function. It gives the expected power as a function of frequency in a long random code string. Several methods of computing the power spectra of codes have been published [7, 12–16]. A recent paper [17] gives a method for computing the power spectra of run-length-limited codes which uses the run-length graph and its associated node and edge probabilities. Explicit formulas for the power spectra of both ideal and implemented RLL codes can be obtained by this method.

The constraint graph and the run-length graph contain all the information necessary for computing the statistical properties of a code, but they do not contain any information about how to map arbitrary data strings into constrained strings and vice versa, i.e., how to encode and decode. Before leaving the subject of MFM, we therefore construct the MFM encoder graph. The encoder graph for a rate m/n code must have three properties:

1. Each node has exactly 2^m outgoing edges.
2. Each edge has an input label and an output label. The input labels on the 2^m outgoing edges from each node are the 2^m binary strings of length m. The output labels are strings of n code symbols.

3. The strings obtained by concatenating the output labels of the edges along any path satisfy the code constraints.

Assume a starting node has been specified in advance. Encoding is done as follows. Take the first m bits from the input string. Select the outgoing edge from the current node whose input label matches the selected m bits. Concatenate the corresponding output label to the encoded string, and make the node to which that edge leads the new current node. Move to the next m-bit block of input bits, and repeat until the input string is exhausted.

The encoder graph can be constructed from the constraint graph. First we construct the nth power of the constraint graph. This is the graph whose adjacency matrix is the nth power of the adjacency matrix of the constraint graph. It has an edge for each path of length n in the constraint graph, and the label on that edge is the concatenation of the labels on the edges making up the path. For the case of MFM, $m = 1$ and $n = 2$. The second power of the MFM constraint graph is shown in **Figure 7**. The squared constraint graph has two connected components because the constraint graph was cyclic of index 2. We may choose either one from which to construct the encoder graph. Choosing the one with two nodes, we see that it has all the required properties. All we need to do is attach input labels 0 and 1 to the two edges leaving each node. This can be done arbitrarily, but the choice in which the input label on each edge matches the second symbol of the output label is a particularly good one. It makes decoding trivial. The final MFM encoder graph is shown in **Figure 8**. The edge label 1/**01** means that 1 is the input label and **01** is the output label. The operations of squaring the graph and discarding one component have destroyed the reversal symmetry of the code when the code is viewed as a sequence of two-symbol blocks. For example, the sequence **00 01** is allowed, but **10 00** is not. After some graph manipulations, the discarded component can be seen to be equivalent to the reversal of the other. It allows **10 00** but not **00 01**.

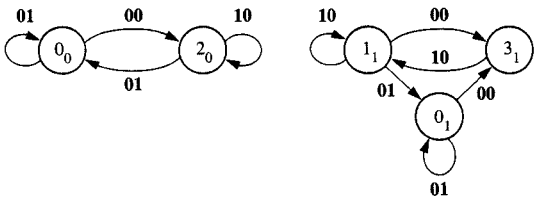


Figure 7

The square of the MFM constraint graph.

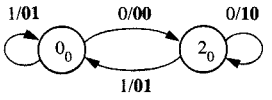


Figure 8

The MFM encoder graph.

• *The (1, 7) code*

The (1, 7) run-length constraints have a capacity of about 0.679. Several authors have published practical (1, 7) codes with a code rate of 2/3 [18–22]. Although these codes have different encoders and decoders, they are all essentially equivalent from the point of view of their statistical properties. The code described in [20] and [21] and the one described in [22] use the same set of constrained strings; in other words, their incidental constraints are the same. They differ only in the mapping of unconstrained user data onto constrained strings; i.e., the encoders and decoders are different. The codes described in [18] and [19] differ from the others in two ways. They use different encoder and decoder mappings, and the data are divided differently into three-symbol blocks. If we label their symbols

$\cdots a_{i-1}b_{i-1}c_{i-1}\, a_ib_ic_i\, a_{i+1}b_{i+1}c_{i+1}\cdots$, then the string $\cdots c_{i-2}a_{i-1}b_{i-1}\, c_{i-1}a_ib_i\, c_ia_{i+1}b_{i+1}\cdots$ is in the other two codes. An observer would find it impossible to determine which of these four codes produced any given string of code symbols. If the symbols were divided into blocks of three, then the strings of three-symbol blocks from the first two codes would be distinguishable from those of the other two. The division into blocks does not affect the statistical properties discussed above, so all four codes have the same statistical properties.

We begin the construction of the 2/3 (1, 7) code by modifying the graph $\mathbf{G}(1, 7)$ to be cyclic of index 3. We do so by taking the Krönecker product of the permutation matrix representing a 3-cycle with $\mathbf{T}(1, 7)$. The resulting matrix is

$$C_3 \otimes \mathbf{T}(1, 7) = \begin{bmatrix} 0 & T(1, 7) & 0 \\ 0 & 0 & T(1, 7) \\ T(1, 7) & 0 & 0 \end{bmatrix}.$$

This is the adjacency matrix of the graph shown in **Figure 9**.

The next step is to introduce the incidental constraints. The incidental constraints for the 2/3 (1, 7) code take the form of constraints on the starting points for runs of six and seven **0**s, modulo 3. Let the ith code symbol be the first **0** in a run of seven. If another run of seven **0**s begins at the jth

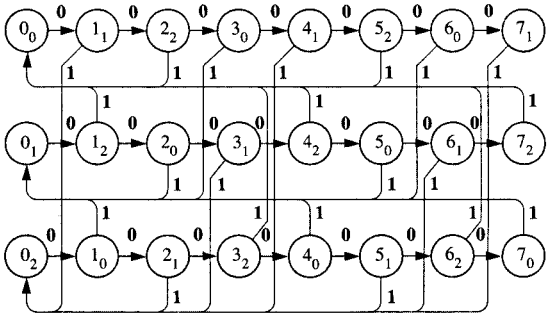


Figure 9

The graph **G**(1, 7) modified to be cyclic of index 3.

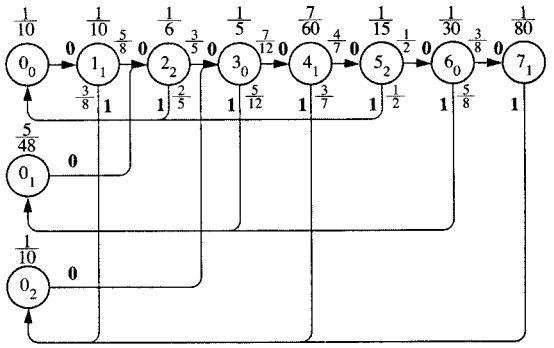


Figure 11

The (1, 7) constraint graph.

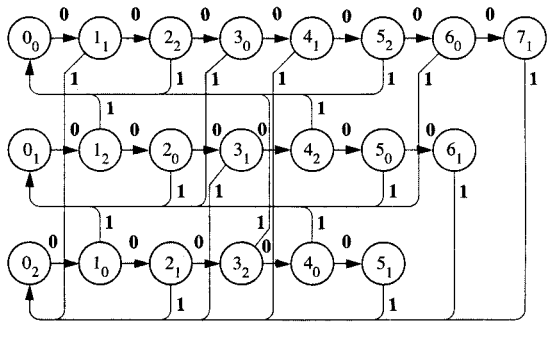


Figure 10

The (1, 7) constraint graph before simplification.

symbol, the incidental constraints require that $i \equiv j \bmod 3$. Similarly, if a run of six **0**s begins at the kth code symbol, the incidental constraints require that $k \equiv i \bmod 3$ or $k \equiv (i+1) \bmod 3$. These constraints, together with the (1, 7) constraints, have capacity 2/3. They can be reflected on the graph by deleting nodes 7_2, 7_0, and 6_2. The resulting graph is shown in **Figure 10**. This graph can be simplified: Nodes 5_1, 6_1, and 7_1 can be combined into one, since they all have the same outgoing edges. Similarly, we can combine the sets $\{4_0, 5_0, 6_0\}$, $\{3_2, 4_2, 5_2\}$, $\{2_1, 3_1, 4_1\}$, $\{1_0, 2_0, 3_0\}$, and $\{1_2, 2_2\}$. **Figure 11** shows the simplified (1, 7) constraint graph with its associated node and edge probabilities. The constraint graph can be used to compute the probabilities of specific strings in the (1, 7) code. Note that all the node and edge probabilities are rational numbers, which means that the probability of any string is also rational. The (1, 7) incidental constraints are symmetrical with respect to reversal, so the reverse of any string has the same probability as the string itself. As an example, we give the probabilities of all the three-symbol strings; the corresponding probabilities for the ideal (1, 7) code are shown for comparison:

$$p(\mathbf{000}) = s_{7_1} + s_{6_0} + s_{5_2} + s_{1_1} p_{1_1 2_2} p_{2_2 3_0} p_{3_0 4_1} + s_{0_1} p_{0_1 2_2} p_{2_2 3_0} p_{3_0 4_1} + s_{0_0} p_{0_0 1_1} p_{1_1 2_2} p_{2_2 3_0}$$

$$= \frac{5}{24} = 0.20833 \quad \text{vs. } 0.23092 \text{ for ideal code,}$$

$$p(\mathbf{001}) = p(\mathbf{100})$$

$$= \frac{11}{60} = 0.18333 \quad \text{vs. } 0.17975 \text{ for ideal code,}$$

$$p(\mathbf{010}) = s_{0_0} + s_{0_1} + s_{0_2}$$

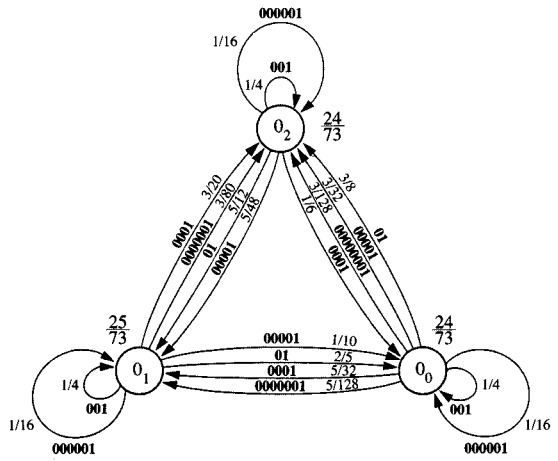
$$= \frac{73}{240} = 0.30417 \quad \text{vs. } 0.29466 \text{ for ideal code,}$$

$$p(\mathbf{100}) = s_{1_1} p_{1_1 2_2} + s_{0_1} p_{0_1 2_2} p_{2_2 3_0} + s_{0_2} p_{0_2 3_0} p_{3_0 4_1}$$

$$= \frac{11}{60} = 0.18333 \quad \text{vs. } 0.17975 \text{ for ideal code,}$$

$$p(\mathbf{101}) = s_{1_1} p_{1_1 0_2} + s_{0_1} p_{0_1 2_2} p_{2_2 0_0} + s_{0_2} p_{0_2 3_0} p_{3_0 0_1}$$

$$= \frac{29}{240} = 0.12083 \quad \text{vs. } 0.11491 \text{ for ideal code.}$$



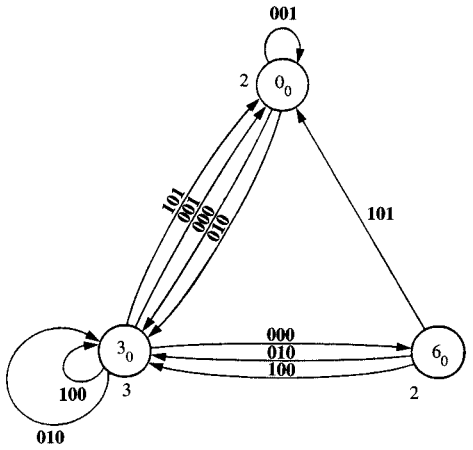


Figure 12

The (1, 7) code run-length graph.

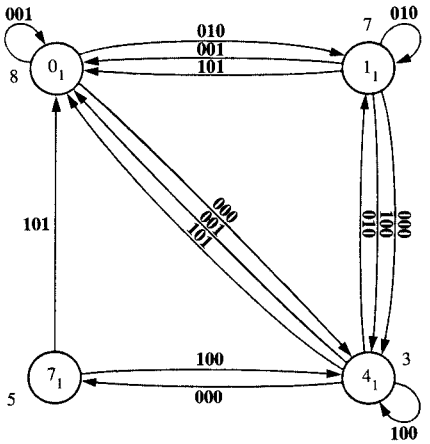
The probability of the symbol **1** is the same as the probability of **010** given above. The average run length (including the **1**) is the reciprocal: (240/73) = 3.28767. The ideal (1, 7) code has an average run length of 3.39374. The run-length graph for the 2/3 (1, 7) code is obtained by projecting the constraint graph onto the three nodes 0_i for $i = 0, 1, 2$. Its node probabilities are obtained by renormalizing the node probabilities from the constraint graph so that they sum to one. The edge probabilities can be computed from the constraint graph; they can also be found by computing the edge probabilities for the reverse of the graph in Figure 10. Each edge in the run-length graph corresponds to a path in the reversed graph. This path contains one edge with probability $q \neq 1$ from 0_i to another node and several edges with probability 1. So the probability of the edge in the run-length graph is also q. The run-length graph is shown in **Figure 12**.

The run-length probabilities from each node can be read from the run-length graph. The overall probability of a given run length is the average of the probabilities of that run length from each node, weighted by the node probabilities. The run-length probabilities for the (1, 7) code are

$$p(\text{run length} = [2\ 3\ 4\ 5\ 6\ 7\ 8])$$

$$= \frac{1}{1168}\,[464\ 292\ 184\ 116\ 73\ 30\ 9].$$

The (1, 7) encoder graph can be constructed from the third power of the constraint graph. There are three connected components, and any one will do. The encoders described in [20] and [22] come from the component $\{0_2, 2_2, 5_2\}$. The encoders in [18] and [19] come from $\{0_0, 3_0, 6_0\}$. **Figure 13** shows the cubed constraint graph. The



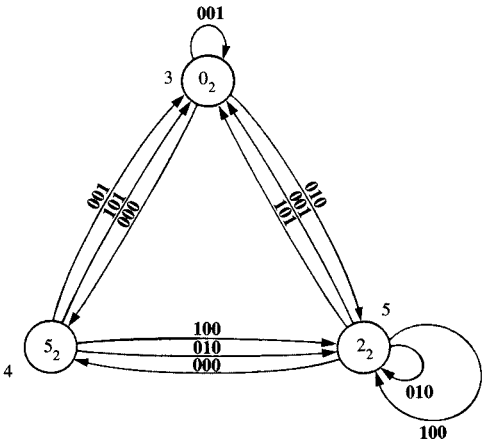


Figure 13

The cube of the (1, 7) constraint graph.

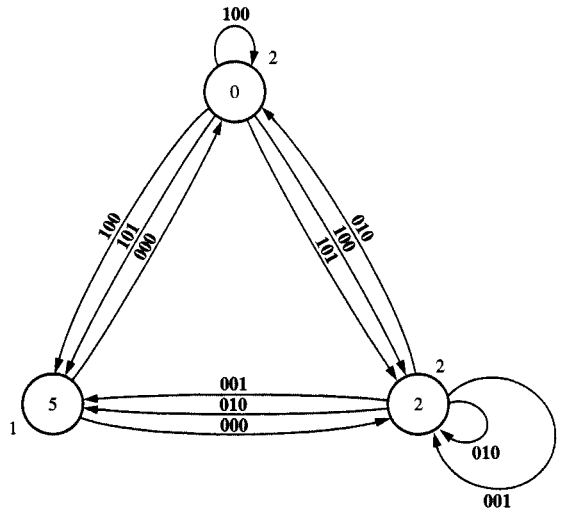


Figure 14

The reverse of the component $\{0_2, 2_2, 5_2\}$ of the cubed (1, 7) constraint graph.

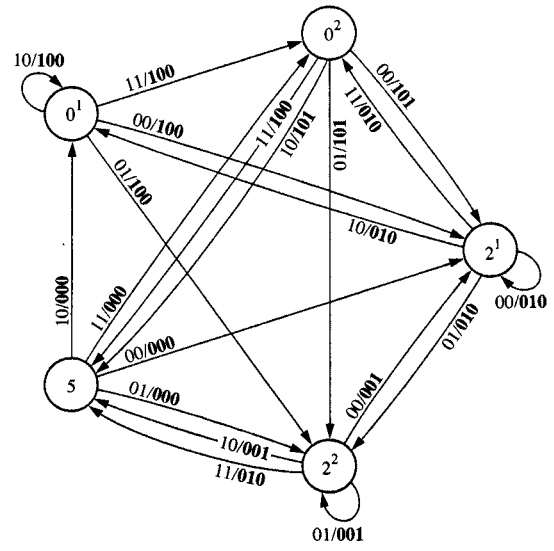


Figure 15

The (1, 7) encoder graph.

integers next to the nodes are components of the eigenvector of the cubed adjacency matrix. They guide the construction of the encoder graph as described in [23]. The sum of the eigenvector components is an upper bound on the number of nodes in the final encoder graph. Since the constraints are symmetrical with respect to reversal, the reverses of these components can also be used to construct encoders. The reverse of the component $\{0_2, 2_2, 5_2\}$ has eigenvector components [2 2 1], so it can be made into a five-node encoder graph by "splitting" the first two nodes. This is simpler than the encoders which would result from any of the components of the cubed constraint graph. **Figure 14** shows the reverse of this component. The subscripts 2 on the node labels have been suppressed for simplicity.

A (1, 7) encoder graph can be constructed by "splitting" the first two nodes. Splitting a node means replacing it with two (or more) nodes. Each new node has the same incoming (outgoing) edges as the original node. The outgoing (incoming) edges from the original node are partitioned among the new nodes. The resulting graph has the same set of path labels as the original graph, so it describes the same set of constrained sequences. However, its nodes have different numbers of edges. Adler et al. [23] describe a method for determining a sequence of splittings which convert a graph into one with the properties of an encoder graph; i.e., all nodes have the same number of outgoing edges. The encoder graph derived from Figure 14 is shown in **Figure 15**. The nodes resulting from splitting node 0 are labeled 0^1 and 0^2, and node 2 is treated similarly. Each node has four outgoing edges to which the four two-bit input labels may be assigned arbitrarily. The input labels shown are the ones used in [22].

• *The (2, 7) code*

The (2, 7) run-length constraints have a capacity of about 0.517. Practical (2, 7) codes have a rate of 1/2. The code used in the IBM 3380 disk storage device is described in [24–26]; different encoders using the same incidental constraints (described below) are given in [23] and [27]. Another (2, 7) code with different incidental constraints is described in [28].

We begin the construction of the 1/2 (2, 7) code by modifying the graph **G**(2, 7) to be cyclic of index 2. We do so by taking the Krönecker product of the permutation matrix representing a 2-cycle with **T**(2, 7). The resulting matrix is

$$C_2 \otimes \mathbf{T}(2, 7) = \begin{bmatrix} 0 & T(2, 7) \\ T(2, 7) & 0 \end{bmatrix}.$$

This is the adjacency matrix of the graph shown in **Figure 16**.

The next step is to introduce the incidental constraints. One of the incidental constraints for the 1/2 (2, 7) code is a constraint on the starting points for runs of seven **0**s, modulo 2. Let the ith code symbol be the first **0** in a run of seven. If another run of seven **0**s begins at the jth symbol,

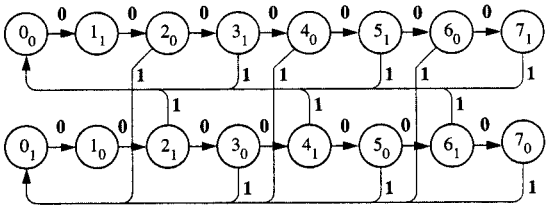


Figure 16

The graph **G**(2, 7) modified to be cyclic of index 2.

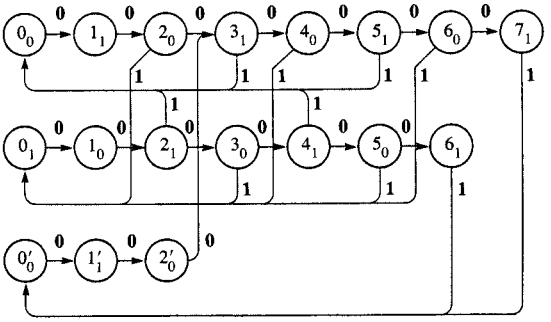


Figure 17

The (2, 7) constraint graph before simplification.

the incidental constraints require that $i \equiv j \bmod 2$. In addition, a run of seven zeros may not be followed immediately by a run of just two zeros. There is a similar constraint on certain runs of six zeros. A run of six zeros starting at index k, where $k \equiv i + 1 \bmod 2$, may not be followed immediately by a run of just two zeros. These constraints, together with the (2, 7) run-length constraints, have capacity 1/2. Unlike the MFM and (1, 7) constraints, the (2, 7) incidental constraints are not symmetrical with respect to reversal. A run of two zeros may not follow runs of seven or certain runs of six zeros, but it may precede them.

The (2, 7) constraint graph is derived from the graph of Figure 16 by deleting node 7_0 and duplicating nodes 0_0, 1_1, and 2_0. The edge from the new copy of 2_0 to 0_1 is deleted. The result is shown in **Figure 17**. This graph can be simplified. Nodes 6_1 and 7_1 can be combined into one, since they have the same outgoing edges. Similarly, we can combine the pairs $\{5_0, 6_0\}$, $\{4_1, 5_1\}$, $\{3_0, 4_0\}$, $\{2_1, 3_1\}$, and $\{1_0, 2_0'\}$. **Figure 18** shows the simplified (2, 7) constraint graph with its associated node and edge probabilities. The constraint graph can be used to compute probabilities of specific strings in the (2, 7) code. All the node and edge probabilities are rational numbers, which means that the probability of any string is also rational. As an example, we give the probabilities of all the four-symbol strings. These probabilities happen to be reversal-symmetric because they are so short, but reversal symmetry fails for long strings. The corresponding probabilities for the ideal (2, 7) code are shown for comparison:

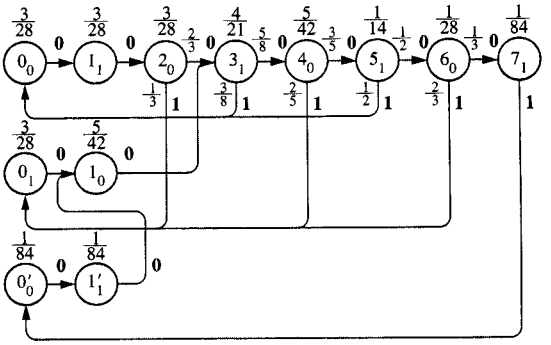


Figure 18

The (2, 7) constraint graph.

$$p(\mathbf{0000}) = \frac{115}{672} = 0.17113 \text{ vs. } 0.19206 \text{ for ideal code,}$$

$$p(\mathbf{0001}) = \frac{101}{672} = 0.15030 \text{ vs. } 0.14551 \text{ for ideal code,}$$

$$p(\mathbf{0010}) = \frac{19}{84} = 0.22619 \text{ vs. } 0.22081 \text{ for ideal code,}$$

$$p(\mathbf{0100}) = \frac{19}{84} = 0.22619 \text{ vs. } 0.22081 \text{ for ideal code,}$$

$$p(\mathbf{1000}) = \frac{101}{672} = 0.15030 \text{ vs. } 0.14551 \text{ for ideal code,}$$

$$p(\mathbf{1001}) = \frac{17}{224} = 0.07589 \text{ vs. } 0.07530 \text{ for ideal code.}$$

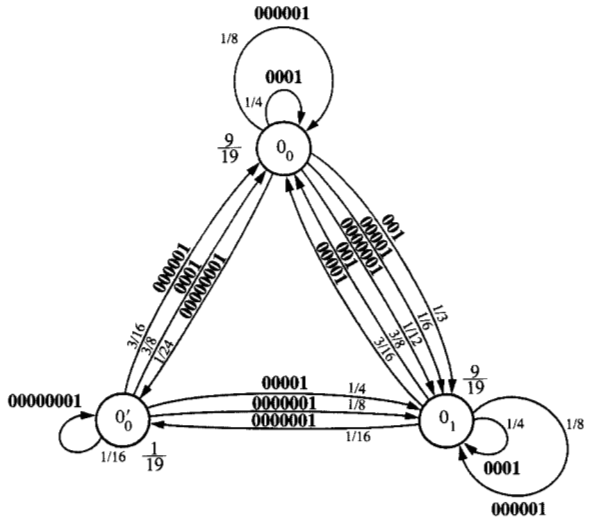


Figure 19

The (2, 7) code run-length graph.

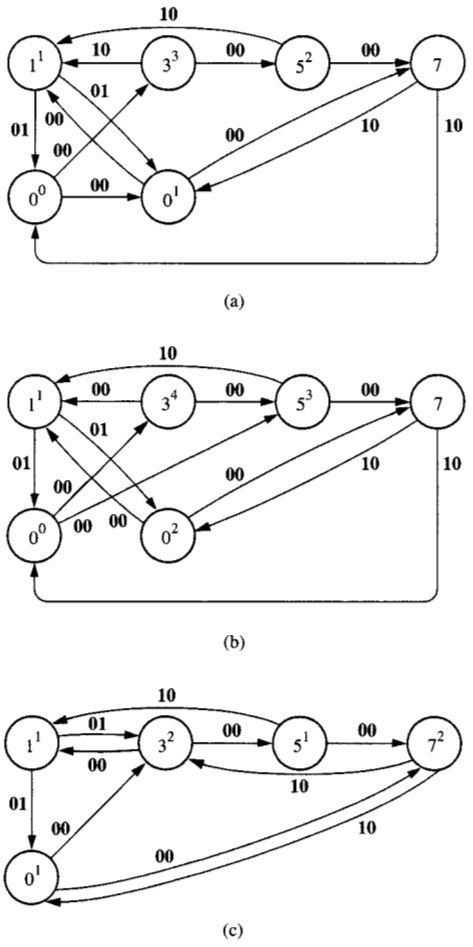
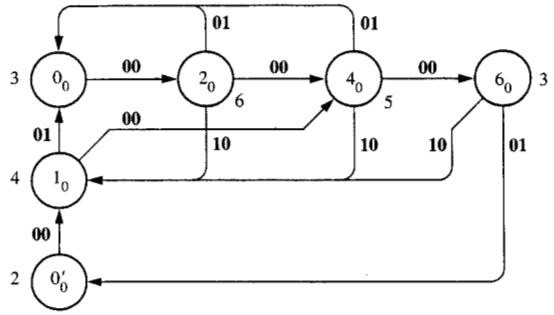


Figure 21

Three possible encoder graphs.



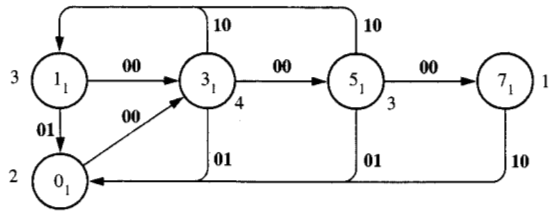


Figure 20

The square of the (2, 7) constraint graph.

Writing expressions for these probabilities in terms of the s_i and p_{ij} can be tedious. The following device makes the process systematic. Let $Q = (q_{ij})$ be written as the sum of two parts: $Q = Q^{\mathbf{0}} + Q^{\mathbf{1}}$, where $Q^{\mathbf{0}}$ contains the contributions from edges labeled **0**, and $Q^{\mathbf{1}}$ contains the contributions from edges labeled **1**. Let **u** be a vector of all ones. Then $p(\mathbf{1001}) = s^{\dagger}Q^{\mathbf{1}}Q^{\mathbf{0}}Q^{\mathbf{0}}Q^{\mathbf{1}}u$, and the probability of any other string can be obtained in a similar way.

The probability of the symbol **1** is the same as the probability of **0010** given above. The average run length (including the **1**) is the reciprocal: (84/19) = 4.42105. The ideal (2, 7) code has an average run length of 4.52879. The run-length graph for the 1/2 (2, 7) code is obtained by projecting the constraint graph onto the three nodes 0_0, 0_1,

and $0_0'$. Its node probabilities are obtained by renormalizing the node probabilities from the constraint graph so that they sum to one. The edge probabilities are computed from the constraint graph. The run-length graph is shown in **Figure 19**.

The run-length probabilities from each node can be read from the run-length graph. The overall probability of a given run length is the average of the probabilities of that run length from each node, weighted by the node probabilities. The run-length probabilities for the (2, 7) code are

$$p(\text{run length} = [3\ 4\ 5\ 6\ 7\ 8]) = \frac{1}{304}[102\ 78\ 55\ 39\ 23\ 7].$$

The constraint graph can be simplified further by combining nodes $\{0_1, 1_1'\}$. This could have been done earlier, but it would have interfered with the construction of the run-length graph. The (2, 7) encoder graph can be constructed from the second power of the constraint graph. There are two connected components, and either one will work. The encoders described in [25–27] come from the component $\{0_1, 1_1, 3_1, 5_1, 7_1\}$. We use this component, too, dropping the subscripts 1 from this point on. **Figure 20** shows the squared constraint graph. The integers next to the nodes are components of the eigenvector of the squared adjacency matrix for the largest eigenvalue, 2. The encoder graph is constructed by splitting states in such a way that the eigenvector components remain integral and the maximum eigenvector component is reduced. The process is complete when all eigenvector components are unity, which means that each state has two outgoing edges. The graph can be simplified by merging pairs of nodes which have identical sets of outgoing or incoming edges. Even with the eigenvector as a guide, there are many ways to complete the construction of the encoder graph. **Figure 21** shows three different graphs which can be derived from the smaller component of Figure 20. Each can be made into an encoder graph by adding input labels.

Figure 22 shows the encoder graph for the (2, 7) code used in several IBM disk products, including the 3380. It is derived from alternative (b) of Figure 21. The advantage of this encoder graph over ones based on alternatives (a) and (c) of Figure 21 is that alternative (b) has smaller maximum error propagation in the decoding process. This can be seen as follows. Consider the received sequence **00 00 10 00**, corresponding to the path in alternative (a) from 0^0 to 0^1 to 7 to 0^1 to 7. A single-bit error could change the received string to **00 00 10 01**. This string corresponds to the path from 0^0 to 3^3 to 5^2 to 1^1 to 0^0. In order to know how to decode the first block **00** we need to look ahead three blocks. Now consider the paths from 0^0 to 3^3 to 1^1 and from 3^3 to 5^2 to 1^1. Either the input labels on the edges $0^0 \rightarrow 3^3$ and $3^3 \rightarrow 5^2$ coincide, or we need to look back one block to decode them. Similarly, the paths from 0^0 to 0^1 to 7 and from 3^3 to 5^2 to 7 require that the input labels on $0^0 \rightarrow 0^1$ and $3^3 \rightarrow 5^2$

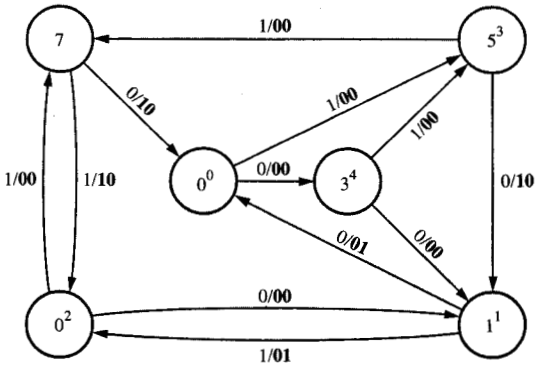


Figure 22

The (2, 7) encoder graph.

coincide, or we need to look back one block to decode them. It is impossible for both pairs to coincide, so we need to look back one block to decode some edge. The decoding of a received block involves five blocks: one previous, three future, and the current block. A single-bit error in the received data could affect the decoding of five blocks, propagating it to five errors in the decoded data. It is easy to see that the decoder for alternative (c) requires four blocks of look-ahead, so it also propagates errors to five bits. The decoder for Figure 22 propagates errors to only four bits.

4. Conclusions

A method has been presented for determining statistical properties of codes. It has been applied to three examples: the MFM, (1, 7), and (2, 7) codes. The method relies on constructing a constraint graph which incorporates all the constraints of the code, both run-length and incidental. The capacity of the constraints for an implemented code must be a rational number, m/n, and this fact gives us useful information for constructing the constraint graph. It requires that the lengths of all cycles in the constraint graph be multiples of n.

Statistics such as the probabilities of specific code strings or the density of **1**s are most easily computed from the constraint graph. Other quantities such as run-length probabilities and power spectra are more easily computed from the run-length graph which is derived from the constraint graph. The mapping of unconstrained input data to code strings is described by the encoder graph. The encoder graph may also be constructed from the constraint graph.

In previous work the usual practice was to derive the encoder graph from the graph $\mathbf{G}(d, k)$, describing only the run-length constraints. Along the way, additional constraints were added implicitly as nodes and edges were discarded. These methods did not lead to an understanding of the incidental constraints, and did not give an easy way to compute the statistical properties of the final code. The main idea of this paper, then, is that it is worthwhile to understand the incidental constraints and to take them into account explicitly when constructing and analyzing a code.

Appendix A: Reversal symmetry

Let c^R denote the reverse of a code string c. Let $\mathbf{G}^R$ denote the reverse of a constraint graph $\mathbf{G}$, produced by reversing the directions of all edges of $\mathbf{G}$ and replacing all edge labels with their reverses. Let $p_{\mathbf{G}}(c)$ denote the probability of the string c in the maxentropic code described by constraint graph $\mathbf{G}$.

Lemma
For any code string c, $p_{\mathbf{G}^R}(c) = p_{\mathbf{G}}(c^R)$.

Proof Consider a path a in $\mathbf{G}^R$ whose label is c. Let the nodes along the path be $s_{i_0}, s_{i_1}, \cdots, s_{i_{k-1}}, s_{i_k}$. The probability for path a in the maxentropic code is

$$p_{\mathbf{G}^R}(a) = s_{i_0} p_{i_0 i_1} \cdots p_{i_{k-1} i_k}$$

$$= \frac{w_{i_0} v_{i_0}}{\sum_i w_i v_i} \frac{w_{i_1}}{\lambda w_{i_0}} \cdots \frac{w_{i_k}}{\lambda w_{i_{k-1}}}$$

$$= \frac{v_{i_0} w_{i_k}}{\lambda^k \sum_i w_i v_i}.$$

Let a^R be the reverse of path a. It is a path in $\mathbf{G}$ passing through nodes $s_{i_k}, s_{i_{k-1}}, \cdots, s_{i_1}, s_{i_0}$. By a similar argument, we find

$$p_{\mathbf{G}}(a^R) = \frac{w_{i_k} v_{i_0}}{\lambda^k \sum_i v_i w_i}.$$

Taking the union over all paths whose labels are equal to c completes the proof of the lemma.

Theorem
Let $\mathbf{G}$ be the constraint graph for a reversal-symmetric code; i.e., $\mathbf{G}$ and $\mathbf{G}^R$ describe the same set of code constraints. The probabilities, $p_{\mathbf{G}}(c)$ and $p_{\mathbf{G}}(c^R)$, of any string c and its reverse are equal.

Proof We have $p_{\mathbf{G}}(c) = p_{\mathbf{G}^R}(c)$ because $\mathbf{G}$ and $\mathbf{G}^R$ describe the same constraints. The maxentropic probabilities do not depend on which description of those constraints is chosen. By the lemma, $p_{\mathbf{G}^R}(c) = p_{\mathbf{G}}(c^R)$. Combining the two equations gives the desired result.

Appendix B: Rational probabilities

Theorem
Let $\mathbf{T}$ be the adjacency matrix for an irreducible graph $\mathbf{G}$ with fixed-length edge labels describing a set of code constraints with rational capacity $C = m/n$. Then the node probabilities s_i and edge probabilities p_{ij} which achieve capacity in $\mathbf{G}$ are rational numbers.

Proof Since $\mathbf{T}$ is nonnegative, it has, by the Perron–Frobenius theorem [29], a positive real eigenvalue λ equal to its spectral radius. Let v and w be the positive real right and left eigenvectors for λ. The following relationships were given in Sections 2 and 3. The first three can be found in [5], and the last is from [8]:

1. $C = \log_2 \lambda$.
2. $p_{ij} = (v_j / \lambda v_i)$ when $t_{ij} \neq 0$, and $p_{ij} = 0$ otherwise.
3. $s_i = (v_i w_i) / \sum_i v_i w_i$.
4. In order to have $C = (m/n)$ with m and n relatively prime, $\mathbf{T}$ must be cyclic of index some multiple of n.

The rest of the proof assumes that $\mathbf{T}$ is cyclic of index n. The proof for the general case is identical with m and n replaced by cm and cn for some integer c. Because $\mathbf{T}$ is cyclic of index n, we may assume it has the following form, after reordering the nodes of $\mathbf{G}$ if necessary [29]:

$$\mathbf{T} = \begin{bmatrix} 0 & T_{1,2} & & & & \\ & 0 & \cdot & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & T_{n-1,n} \\ T_{n,1} & & & & & 0 \end{bmatrix}. \tag{A1}$$

The blocks into which $\mathbf{T}$ is partitioned correspond to the connected components of the nth power of $\mathbf{G}$. Let $v = [v^{(1)} \cdots v^{(n)}]$ and $w = [w^{(1)} \cdots w^{(n)}]$ be similarly partitioned. The vectors $v^{(1)}$ and $w^{(1)}$ satisfy linear equations with integer coefficients: $\mathbf{T}_{1,1}^{(n)} v^{(1)} = 2^m v^{(1)}$ and $w^{(1)\dagger} \mathbf{T}_{1,1}^{(n)} = 2^m w^{(1)\dagger}$, where we have used $\lambda^n = 2^m$, and $\mathbf{T}_{1,1}^{(n)} = \mathbf{T}_{1,2} \mathbf{T}_{2,3} \cdots \mathbf{T}_{n-1,n} \mathbf{T}_{n,1}$. Therefore, we may normalize the vectors v and w such that $v^{(1)}$ and $w^{(1)}$ have rational (or integer) components.

The equation $\mathbf{T}_{1,2} v^{(2)} = \lambda v^{(1)}$ implies that $v^{(2)}/\lambda$ has rational entries. The equation $\mathbf{T}_{k-1,k} v^{(k)} = \lambda v^{(k-1)}$ establishes by induction that $v^{(k)}/\lambda^{k-1}$ has rational entries for $1 \leq k \leq n$. Similarly, the equation $w^{(k)\dagger} \mathbf{T}_{k,k+1} = \lambda w^{(k+1)\dagger}$ shows that $w^{(k)} \lambda^{k-1}$ is rational for $1 \leq k \leq n$.

The representation of $\mathbf{T}$ in (A1) shows that every edge in $\mathbf{G}$ goes from a node i in component (k) to a node j in component $(k+1)$, where the component numbers are treated cyclically: Component $(n+1)$ is component (1). The edge probability is

$$p_{ij} = \frac{v_j}{\lambda v_i} = \frac{r_1 \lambda^k}{\lambda r_2 \lambda^{k-1}},$$

where r_1 and r_2 are rational numbers. This establishes that the edge probabilities are rational.

The node probability is $s_i = (v_i w_i)/\sum_i v_i w_i$. Let node i be in component (k). The product $v_i w_i = r_1 \lambda^{k-1} r_2 \lambda^{-(k-1)}$, where r_1 and r_2 are rational numbers, is rational for all i. It follows that the node probabilities are rational, and the proof is complete.

The node and edge probabilities in the run-length graph are rational functions of those for the constraint graph, so they are rational as well. The requirement for fixed-length edge labels can be removed. A graph with variable-length edge labels can be converted to an equivalent graph with fixed-length edge labels by replacing each edge having a multi-symbol label by a sequence of edges with single-symbol labels. The theorem applies to the new graph, and rational operations on its probabilities yield the node and edge probabilities for the original graph. The generalization of the theorem to constraints with rational b-ary capacity, where b is an integer, is obvious.

An alternative proof of the theorem can be obtained by inverting the sequence of node splittings and mergings by which an encoder graph is derived from the constraint graph. The existence of a suitable sequence of splittings and mergings is guaranteed by a theorem of Adler et al. [23]. The encoder graph has edge probabilities 2^{-m}, and its node probabilities are rational because they satisfy $s^{\dagger}(Q - I) = 0$. The proof is completed by showing that node mergings and splittings change the node and edge probabilities by rational operations.

References

1. W. Kautz, "Fibonacci Codes for Synchronization Control," *IEEE Trans. Info. Theory* **IT-11,** 284–292 (1965).
2. P. Franaszek, "Sequence-State Methods for Run-Length-Limited Coding," *IBM J. Res. Develop.* **14,** 376–383 (1970).
3. P. Siegel, "Recording Codes for Digital Magnetic Storage," *IEEE Trans. Magnetics* **MAG-21,** 1344–1349 (1985).
4. K. Schouhamer Immink, "Coding Methods for High Density Optical Recording," *Philips J. Res.* **41,** 410–430 (1986).
5. C. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.* **27,** 379–423 and 623–656 (1948).
6. K. Schouhamer Immink, "Some Statistical Properties of Maxentropic Runlength-Limited Sequences," *Philips J. Res.* **38,** 138–149 (1983).
7. E. Zehavi and J. Wolf, "On Run-Length Limited Codes," *IEEE Trans. Info. Theory* **34,** 45–54 (1988).
8. J. Ashley and P. Siegel, "A Note on the Capacity of Run-Length-Limited Codes," *IEEE Trans. Info. Theory* **IT-33,** 601–605 (1987).
9. P. Siegel, "Applications of a Peak Detection Channel Model," *IEEE Trans. Magnetics* **MAG-18,** 1250–1252 (1982).
10. P. Shaft, "Bandwidth Compaction Codes for Communication," *IEEE Trans. Commun.* **COM-21,** 687–695 (1973).
11. D. Tang and L. Bahl, "Block Codes for a Class of Constrained Noiseless Channels," *Info. Control* **17,** 436–461 (1970).
12. M. Hecht and A. Guida, "Delay Modulation," *Proc. IEEE* **57,** 1314–1316 (1969).
13. J. Lawson and G. Uhlenbeck, *Threshold Signals*, Boston Technical Publishers, Lexington, MA, 1964, pp. 42–46.
14. G. Cariolaro and G. Tronca, "Spectra of Block Coded Digital Signals," *IEEE Trans. Commun.* **COM-22,** 1555–1564 (1974).
15. M. Pelchat and J. Geist, "Surprising Properties of Two-Level 'Bandwidth Compaction' Codes," *IEEE Trans. Commun.* **COM-23,** 878–883 (1975).
16. D. Lindholm, "Power Spectra of Channel Codes for Digital Magnetic Recording," *IEEE Trans. Magnetics* **MAG-14,** 321–323 (1978).
17. A. Gallopoulos, C. Heegard, and P. Siegel, "The Power Spectrum of Run-Length-Limited Codes," *IEEE Trans. Commun.*, in press.
18. T. Horiguchi and K. Morita, "An Optimization of Modulation Codes in Digital Recording," *IEEE Trans. Magnetics* **MAG-12,** 740–742 (1976).
19. P. Franaszek, "Efficient Code for Digital Magnetic Recording," *IBM Tech. Disclosure Bull.* **23,** 4375–4378 (1981).
20. M. Cohn, G. Jacoby, and A. Bates III, "Data Encoding Method and System Employing Two-Thirds Rate Code with Full Word Look-Ahead," U.S. Patent 4,337,458, 1982.
21. G. Jacoby and R. Kost, "Binary Two-Thirds Rate Code with Full Word Look-Ahead," *IEEE Trans. Magnetics* **MAG-20,** 709–714 (1984).
22. R. Adler, M. Hassner, and J. Moussouris, "Method and Apparatus for Generating a Noiseless Sliding Block Code for a (1, 7) Channel with Rate 2/3," U.S. Patent 4,413,251, 1982.
23. R. Adler, D. Coppersmith, and M. Hassner, "Algorithms for Sliding Block Codes," *IEEE Trans. Info. Theory* **IT-29,** 5–22 (1983).
24. P. Franaszek, "Run-Length-Limited Variable Length Coding with Error Propagation Limitation," U.S. Patent 3,689,899, 1972.
25. J. Eggenberger and P. Hodges, "Sequential Encoding and Decoding of Variable Length, Fixed Rate Data Codes," U.S. Patent 4,115,768, 1978.
26. T. Howell, "Analysis of Correctable Errors in the IBM 3380 Disk File," *IBM J. Res. Develop.* **28,** 206–211 (1984).
27. K. Norris, "Run-Length-Limited Codes," *Xerox Disclosure J.* **5,** 647–648 (1980).
28. M. Cohn and G. Jacoby, "Run-Length Reduction of 3PM Code via Look-Ahead Technique," *IEEE Trans. Magnetics* **MAG-18,** 1253–1255 (1982).
29. R. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1962, pp. 30–39.

Received August 1, 1988; accepted for publication December 15, 1988

Thomas D. Howell *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120.* Dr. Howell is a Research Staff Member at the Almaden Research Center. He is currently working on signal processing and coding for magnetic recording channels. Dr. Howell received his B.S. in mathematics from the California Institute of Technology, Pasadena, in 1973, his M.S. in computer science from Cornell University in 1975, and his Ph.D. in computer science in 1976, also from Cornell University. From 1976 to 1977, he was an IBM Postdoctoral Fellow at the University of California at Berkeley. Dr. Howell joined IBM at the San Jose Research Laboratory in 1977, and worked until 1979 on a performance evaluation tool for computer systems called the VM Emulator. He joined the recording channel project at San Jose Research in 1979 and was its manager from 1979 until 1983. Dr. Howell took a one-year assignment at the IBM Zurich Research Laboratory in 1983–84. He received an IBM Outstanding Technical Achievement Award in 1983 for his work on recording channel modeling. Dr. Howell is a member of the Association for Computing Machinery, the Institute of Electrical and Electronics Engineers, and the Mathematical Association of America.