Error-Correction Coding for Digital Communications

George C. Clark, Jr.
and
J. Bibb Cain

Harris Corporation
Melbourne, Florida

PLENUM PRESS • NEW YORK AND LONDON

Error-Correction Coding for Digital Communications

George C. Clark, Jr.
and
J. Bibb Cain

Harris Corporation
Melbourne, Florida

PLENUM PRESS • NEW YORK AND LONDON

Contents

1. Fundamental Concepts of Coding . . . 1

1.1. Basic Principles . . . 2
1.2. Practical Constraints . . . 8
1.2.1. Data Source . . . 9
1.2.2. Encoder . . . 9
1.2.3. Modulator . . . 10
1.2.4. Waveform Channel . . . 11
1.2.5. Demodulator . . . 12
1.2.6. Decoder . . . 14
1.2.7. Discrete Data Channel . . . 15
1.3. Performance Computations . . . 18
1.3.1. Exact Computation of Error Rate . . . 18
1.3.2. Union Bound Computation of Error Rate . . . 23
1.3.3. Performance of Error Detection Systems . . . 24
1.3.4. Soft Decision Decoding . . . 26
1.3.5. Quantization Effects . . . 30
1.3.6. Effect of Noncoherent Combining . . . 34
1.3.7. Coding Gain . . . 35
1.4. Coding Bounds . . . 37
1.4.1. Bounds on Minimum Distance . . . 38
1.4.2. Random Coding Bound . . . 41
1.5. Remarks . . . 46
Problems . . . 47

2. Group Codes . . . 49

2.1. Generalized Parity Check Codes . . . 50
2.1.1. Parity Check Matrix . . . 52
2.1.2. Minimum Distance Considerations . . . 53
2.1.3. Hamming Codes . . . 54

2.1.4. Generator Matrix 56
2.1.5. Dual Codes 57
2.1.6. The Syndrome 58
2.1.7. The Penny-Weighing Problem 59
2.1.8. A Simple Iterated Code 61
2.2. Polynomial Codes 62
2.2.1. Finite Field Arithmetic 63
2.2.2. Generation of Polynomial Codes 68
2.2.3. Cyclic Codes 72
2.2.4. Alternate Encoder for Cyclic Codes 73
2.2.5. Additional Properties of Polynomials and Galois Field Elements 75
2.2.6. Codes Specified by Roots 78
2.2.7. Syndrome Polynomials 79
2.2.8. Modifications of Codes 84
2.3. Important Classes of Group Codes 85
2.3.1. BCH Codes 86
2.3.2. Golay Codes 88
2.3.3. Maximal-Length Codes 89
2.3.4. Reed–Muller Codes 91
2.3.5. Quadratic Residue Codes 92
2.3.6. Remarks 94
Problems 94

3. Simple Nonalgebraic Decoding Techniques for Group Codes 97

3.1. Meggitt Decoders 97
3.2. Information Set Decoding 102
3.2.1. General Decoding Algorithm 102
3.2.2. Information Set Decoding Using the **H** Matrix 107
3.2.3. Predetermined Covering Sets 109
3.2.4. Random Search Techniques 119
3.3. Threshold Decoding 131
3.4. Remarks 137
Problems 138

4. Soft Decision Decoding of Block Codes 141

4.1. APP Threshold Decoding 143
4.1.1. Derivation of the APP Decoding Rule 144
4.1.2. Computation of the Weighting Terms 146
4.1.3. Approximate Computation of w_i 148
4.1.4. Implementation of the APP Decoder 150
4.2. Optimum Symbol-by-Symbol Decoding 154
4.2.1. Derivation of the Hartmann–Rudolph Algorithm 155
4.2.2. An Alternate Form of the HR Algorithm 156
4.2.3. A Design Example 158
4.2.4. Greenberger Approximation to the HR Algorithm 161

4.3. The Weldon Algorithm 162
4.4. The Chase Algorithm 167
4.4.1. Standard Chase Algorithms 167
4.4.2. Variations on the Chase Algorithms 169
4.5. Information-Set-Decoding Algorithms 172
4.5.1. Information-Set-Only Algorithms 172
4.5.2. Omura-Type Algorithms 175
4.5.3. Partial Syndrome Decoding 177
4.5.4. Some Performance Predictions 178
Problems 179

5. Algebraic Techniques for Multiple Error Correction 181

5.1. Finite-Field Transforms 183
5.2. BCH Codes 185
5.3. BCH Decoding Techniques 188
5.3.1. Frequency Domain Decoding 188
5.3.2. Time Domain Decoding 191
5.4. Solution of the Key Equation 195
5.4.1. Euclid's Algorithm 195
5.4.2. Berlekamp's Algorithm 201
5.5. Implementation Considerations 209
5.6. Correction of Errors and Erasures 214
5.7. Performance Results 219
5.8. Remarks 222
Problems 224

6. Convolutional Code Structure and Viterbi Decoding 227

6.1. Binary Rate-1/2 Convolutional Codes 228
6.2. Viterbi Decoding Algorithm 231
6.3. Rate-m/n Convolutional Codes 235
6.4. Finite-State Machine Description and Distance Properties of Convolutional Codes 238
6.5. Performance of Convolutional Codes with Viterbi Decoding 242
6.5.1. Union Bounds 243
6.5.2. Optimum Codes 246
6.5.3. Performance with Coherent PSK 247
6.5.4. Performance with Orthogonal Signaling and Noncoherent Detection 250
6.6. Implementation Considerations 253
6.6.1. Synchronizer 255
6.6.2. Branch Metric Computer 256
6.6.3. Path Metric Storage and Updating 258
6.6.4. Storage and Updating of Hypothesized Information Sequences 261
6.6.5. Output Device 263
6.6.6. Demodulator Quantization and AGC 264
6.7. Remarks 265
Problems 266

7. Other Convolutional Decoding Techniques 267
7.1. Syndrome Decoding Techniques 268
7.1.1. Basic Concepts 268
7.1.2. Table Look-up Decoding 278
7.1.3. Threshold Decoding 283
7.2. Sequential Decoding Techniques 297
7.2.1. Sequential Decoding via the Fano Algorithm 299
7.2.2. Selection of Sequential Decoding Metric 302
7.2.3. Code Selection 304
7.2.4. Sequential Decoder Computational Problem 304
7.2.5. Performance of Sequential Decoders 313
7.2.6. Implementation Considerations 318
7.2.7. Stack Sequential Decoding 326
Problems 328

8. System Applications 331
8.1. Concatenated Codes 331
8.1.1. Concatenated Coding Concepts 332
8.1.2. Reed–Solomon/Orthogonal Code Systems 333
8.1.3. Reed–Solomon/Short-Block Code Systems 337
8.1.4. Reed–Solomon/Convolutional Code Systems 339
8.2. Coding for the White Gaussian Noise Channel 341
8.3. Interleaver Structures for Coded Systems 345
8.3.1. Periodic Interleavers 346
8.3.2. Pseudorandom Interleavers 349
8.4. Coding for Burst Noise Channels 352
8.4.1. Burst Noise Processes 353
8.4.2. Convolutional Code Performance with Random Erasures . . 354
8.4.3. Convolutional Code Performance with Periodic Burst Erasures 358
8.4.4. Performance Degradation Due to Random Erasure Burst . . 363
8.4.5. Interleaving Implications 364
8.5. Coding for Spread Spectrum Systems 366
8.5.1. Direct Sequence PN Spread Systems 367
8.5.2. Frequency Hop Systems 375
8.6. Coding for Bandwidth-Constrained Channels 381
Problems 390

Appendix A. Code Generators for BCH Codes 393
Appendix B. Code Generators for Convolutional Codes 399
B.1. Viterbi Decoding 399
B.2. Table Look-up Decoding 400
B.3. Threshold Decoding 400
B.4. Sequential Decoding 401

References 409
Index 415

Error-Correction Coding for Digital Communications

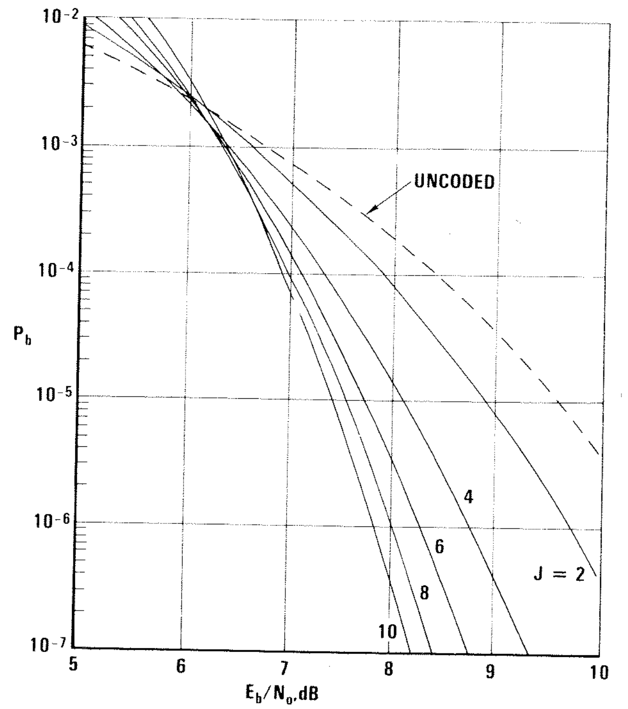

Figure 7-10. Performance of $R = 2/3$, self-orthogonal codes with hard decisions and definite decoding.

hard decisions (though it is much more tedious). This approach was used to calculate the performance curves of Fig. 7-11 for four-level quantization. Since the codes used are CSOCs, $P_b = P_{fe}$. The quantizer level was set at $0.4E_s^{1/2}$. Deviations of ± 3 dB in this setting cause only about 0.1 dB of performance degradation. The additional gain obtained by using four-level rather than two-level quantization is about 0.9 dB. A similar gain results for the higher-rate codes. It appears that the additional gain available with soft decisions using approximate APP decoding decreases as the size of each estimate increases [which is equal to $J(n - 1)$ for these codes].

Significantly more coding gain is obtained by using feedback decoding. For example, by using feedback with a $R = 1/2$, $J = 6$ code, the parity check sizes can be reduced from $\{6, 6, 6, 6, 6, 6\}$ to $\{1, 2, 3, 4, 5, 6\}$. With the reduced sizes, the probability of first decoding error, P_{fe}, can be calculated in the same fashion. This quantity is substantially less than P_b for definite decoding. Typically, P_b for feedback decoding is 2 to 3 times P_{fe}, but it must be determined through simulation. For most codes of interest, the use of feedback decoding has been shown through simulation to provide

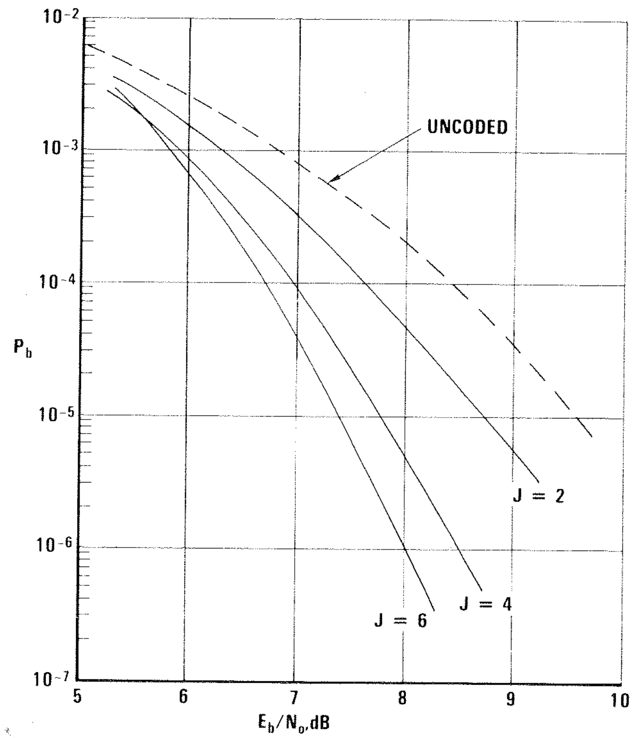

Figure 7-11. Performance of $R = 1/2$ CSOCs with four-level quantization and definite decoding.

about 0.6 dB more gain than definite decoding (at $P_b = 10^{-5}$) both for hard and soft decisions.

Performance calculations indicate that if J is kept fixed and code rate is varied, peak coding gain typically occurs for high code rates ($3/4 < R < 5/6$). The peak gain can be roughly 0.5 dB higher than that achieved with $R = 1/2$ codes. Unfortunately, the decoder complexity increases as n^2 so that increasing n significantly to achieve greater gain is not cost effective. The principal advantage of the higher-rate CSOC codes is that they can offer moderate coding gain in applications where only a very limited amount of redundancy is allowable.

7.2. Sequential Decoding Techniques

Sequential decoding was first introduced by Wozencraft,[64] but the most widely used algorithm to date is due to Fano.[74] Our discussions will focus principally on the Fano algorithm, but subsequently some comparisons with the "stack" sequential decoding algorithm[75,76] will be made.

While a Viterbi decoder extends and updates the metrics for all paths that can potentially be the best, a sequential decoder severely limits the number of paths that are actually updated. The fundamental idea of sequential decoding is that only the path that "appears" to be the most probable should be extended. We say "appears" since, because of the limited search, the decoder is never completely certain that this path is the best. This approach can be viewed as a trial-and-error technique for searching out the correct path in the code tree. It performs the search in a sequential manner always operating only on a single path. However, the decoder is allowed to back up and change previous decisions. Each time the decoder moves forward a "tentative" decision is made. The decision is made by extending the path being examined along its most probable branch. If an incorrect decision is made, subsequent extensions of the path will be wrong. The decoder is eventually able to recognize this situation by examining the path metric. However, it is quite possible to extend an incorrect path for many branches. When this occurs, a substantial amount of computation is required to recover the correct path. The decoder must search back and try alternate paths until it finally decodes successfully. It is very important to carefully select decoder parameters to allow quick recognition of an incorrect decision and quick recovery of the correct paths in order to mitigate the computational problem. The primary benefit of the sequential decoding approach is that each correct decision contributes to limiting the amount of computation that must be performed subsequently. At the same time, the path metric is providing an indication of the correctness of earlier decisions.

The branch metric at node i is defined by

$$\lambda_i = \sum_{j=1}^{n} \left\{ \log_2 \left[\frac{P(r_i^j \mid t_i^j)}{P(r_i^j)} \right] - B \right\} \tag{7-52}$$

where t_i^j is the transmitted symbol and r_i^j is the received symbol. (A binary $R = m/n$ code with n symbols per branch is assumed.) The path metric through node k is simply

$$L_k = \sum_{i=0}^{k} \lambda_i \tag{7-53}$$

The parameter B is a bias term to be explained later. Generally, this term is chosen such that the metric will be increasing in value over a correct path and decreasing in value over an incorrect path. The typical metric behavior is shown in Fig. 7-12. Although the metric for the correct path may temporarily show large decreases due to channel noise, over a longer

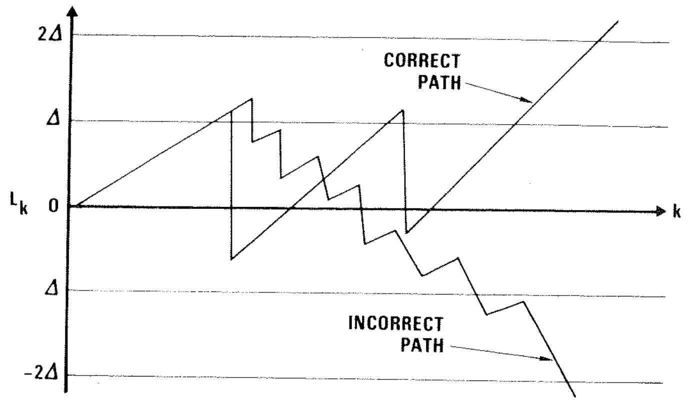

Figure 7-12. Typical metric behavior of correct and incorrect paths.

period of time it should be an increasing function. Also, if a burst of channel noise occurs, the metric on an incorrect path may increase temporarily making the path look like a good path, but it will usually start decreasing when the noise subsides. A sequential decoding algorithm must be able to detect a decreasing metric and quickly find a path with an increasing metric. To do this it uses a *running threshold* T which may be raised or lowered by increments of Δ, the *threshold spacing*. When the current metric falls below the running threshold T, it is an indication that the path being followed may be bad, and a search for a good path is initiated.

7.2.1. Sequential Decoding via the Fano Algorithm

The operation of a sequential decoder is just a trial-and-error search. The principal decoding rule for the Fano algorithm is that the decoder will never proceed either forward or backward on a path that falls below the current threshold. The 2^m branches emanating from the node under consideration are compared to the received branch and ordered according to their likelihood of occurrence. Normally, the decoder moves forward on the most likely branch at each node as long as the metric lies above the current threshold. Also, if $T + \Delta$ is crossed in the positive direction, the threshold is raised by Δ. When the decoder cannot move forward, it attempts to move back without violating the threshold and search all paths that have not been tried with the current threshold. When no paths can be found that satisfy the threshold, the threshold is lowered by Δ and the decoder begins searching forward again. A set of rules for implementing this procedure is shown in Table 7-1. This algorithm, due to Gallager,[3] is a restatement of Fano's sequential decoding algorithm.

Table 7-1. Sequential Decoding Rules for the Fano Algorithm

	Conditions		Action	
Rule	Previous move	Comparisons[a]	Final threshold	Move
1	F or L	$L_{k-1} < T + \Delta$, $L_k \geq T$	Raise (if possible)	F
2	F or L	$L_{k-1} \geq T + \Delta$, $L_k \geq T$	No change	F
3	F or L	any L_{k-1}, $L_k < T$	No change	B
4	B	$L_{k-1} < T$, any L_k	Lower	F
5	B	$L_{k-1} \geq T$, any L_k	No change	L or B

[a] By convention set $L_0 = 0$ and $L_{-1} = -\infty$.

The letters F, L, and B in Table 7-1 denote forward, lateral, and backward moves by the decoder. During a forward move the decoder moves forward to the most likely of the 2^m nodes emanating from the current node. A backward move is simply a move to the node preceding the current node. A lateral move is a move to a node differing from the current node only in the final branch and being the next most likely node in the order of their likelihood. When there is little noise, the decoder is normally using rule 1 and following the correct path. The decoder can proceed for many branches making only forward moves and extending only a single path.

The threshold is raised whenever possible to keep it less than Δ below L_k. If the noise becomes more severe and the metric falls below the threshold after a forward move, rule 3 is invoked and the decoder moves backward in order to find a path that lies above the current threshold. After a backward move either rule 4 or rule 5 is invoked depending upon whether L_{k-1} lies below or above the current threshold. If $L_{k-1} \geq T$, then a lateral move to the next most likely node is made if such a node exists, and a backward move is made if such a node does not exist. If $L_{k-1} < T$, no further backward moves are possible without violating the current threshold so the threshold is lowered by Δ and a forward move is made. The decoder now tries to find a path remaining above this reduced threshold. Rule 2 will now be used. As long as $L_{k-1} \geq T + \Delta$, the threshold cannot be raised, because if it were, the decoder would become trapped in an endless loop.

To better understand the Fano algorithm, consider the following simple example based on the $R = 1/2$, $v = 1$ code with the code tree shown in Fig. 7-1. Assume for the moment that hard decision decoding is used, and that the transmitted information sequence is 1 0 0 0 0.... The received sequence is $\mathbf{R} = 01\ 01\ 00\ 01\ 00\ 00\ldots$. That is, there are single errors in the first and fourth branches. The branch metrics corresponding to this code with the received sequence $\mathbf{R}$ are shown in the tree of Fig. 7-13 using metric increments of $+0.5$ for each symbol that agrees and -4.5 for each

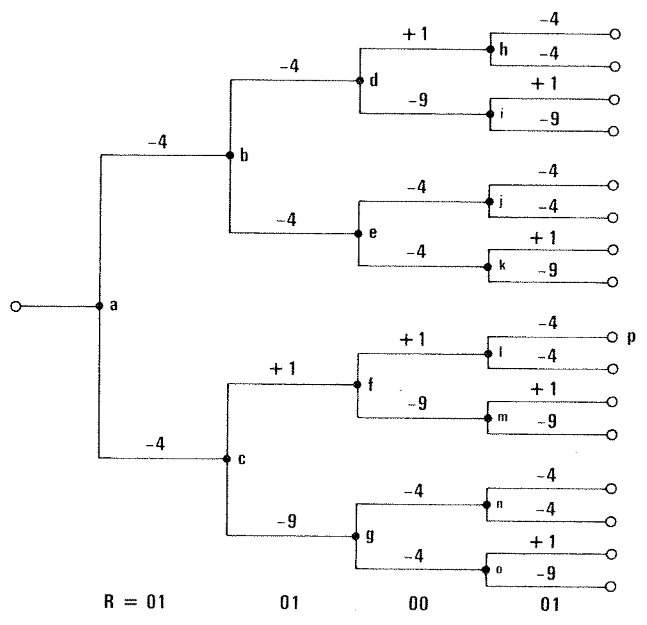

Figure 7-13. Branch metrics for the tree of Fig. 7-1.

Table 7-2. Tree Search of the Fano Algorithm for the Example of Fig. 7-13

Node	T	k	L_k	Final T	Move
b	0	1	−4	0	B
a	0	0	0	−5	F
b	−5	1	−4	−5	F
d	−5	2	−8	−5	B
b	−5	1	−4	−5	L
c	−5	1	−4	−5	F
f	−5	2	−3	−5	F
l	−5	3	−2	−5	F
p	−5	4	−6	−5	B
l	−5	3	−2	−5	B
f	−5	2	−3	−5	B
c	−5	1	−4	−5	B
a	−5	0	0	−10	F

symbol that disagrees. (The selection of these values for metric increments will be justified subsequently.) The steps taken by the Fano algorithm are listed in Table 7-2. The convention used when the decoder is presented with two branches with the same metric (as in the first step) is that the upper branch will always be tried first. Thus, the algorithm begins by trying to find a good path in the upper half tree. The first step forward results in a threshold failure which forces a step back and lowering of the threshold. However, in proceeding forward in the same direction to node *d*, another threshold failure occurs. This forces a movement back and into the lower half tree in an attempt to find a better path. Forward movement along the best path continues until node *p*, where another channel error causes a threshold failure. The decoder then must retrace its steps until it reaches node *a* and lowers the threshold to -10. Now there is a single path that always lies above this threshold. After searching the upper half tree, the decoder can move forward along the path *a–c–f–l–p* without violating the threshold. (These steps are left as an exercise for the reader.) The correct path has been found, and since there are no further channel errors, the metric will continually increase from this node along the correct path and only forward moves will be made.

7.2.2. Selection of Sequential Decoding Metric

The sequential decoding metric of (7-52) with $B = R$ (the code rate) was suggested by Fano based on heuristic arguments. Massey[77] has demonstrated that this metric allows the decoder to extend the most likely path based on the information available at each iteration of the decoding algorithm. These arguments will be briefly presented to provide further insight into sequential decoder behavior.

Consider a $R = m/n$ binary convolutional code and a sequential decoder which is attempting to decode the vector

$$\mathbf{r}_N = (r_0^1, \ldots, r_0^n, r_1^1, \ldots, r_1^n, \ldots, r_N^1, \ldots, r_N^n)$$

of the first $N + 1$ received branches. At any point the decoder will have examined M potential code tree sequences of varying lengths and compared them with $\mathbf{r}_N$. These sequences $\{\mathbf{t}_{k_1}, \mathbf{t}_{k_2}, \ldots, \mathbf{t}_{k_M}\}$ are each of the form

$$\mathbf{t}_k = (t_0^1, \ldots, t_0^n, t_1^1, \ldots, t_1^n, \ldots, t_k^1, \ldots, t_k^n)$$

A sequential decoder is restricted to extending one of these M sequences on its next move, and it is assumed to have no knowledge of the sequences in the unexplored part of the code tree. Thus, we shall select as the path $\mathbf{t}_k$ to be extended that path which maximizes $\Pr(\mathbf{t}_k | \mathbf{r}_N)$. Note that by Bayes' rule

$$\Pr(\mathbf{t}_k | \mathbf{r}_N) = \frac{\Pr(\mathbf{r}_N | \mathbf{t}_k) \Pr(\mathbf{t}_k)}{\Pr(\mathbf{r}_N)}$$

and this is equivalent to maximizing

$$\Pr(\mathbf{r}_N | \mathbf{t}_k) \Pr(\mathbf{t}_k)$$

or to maximizing

$$\Pr(\mathbf{r}_N | \mathbf{t}_k) \Pr(\mathbf{t}_k) \Big/ \prod_{i=0}^{N} \prod_{j=1}^{n} \Pr(r_i^j)$$

Assuming information symbols that are independent and equally likely to be zeros or ones we have

$$\Pr(\mathbf{t}_k) = 2^{-(k+1)m}$$

In addition, since we have assumed that the decoder has no knowledge of the extensions of path $\mathbf{t}_k$ beyond the kth node, we model this situation as if the path $\mathbf{t}_k$ had a random tail added increasing its length to N nodes. Thus, we write

$$\Pr(\mathbf{r}_N | \mathbf{t}_k) = \prod_{i=0}^{k} \prod_{j=1}^{n} \Pr(r_i^j | t_i^j) \prod_{i=k+1}^{N} \prod_{j=1}^{n} \Pr(r_i^j)$$

Using these expressions we note that the decoder should maximize

$$2^{-(k+1)m} \prod_{i=0}^{k} \prod_{j=1}^{n} \frac{\Pr(r_i^j | t_i^j)}{\Pr(r_i^j)}$$

Finally, taking $\log_2(\cdot)$ of this expression gives (7-52) and (7-53) with $B = R$.

This result states that given the information that is available ($\mathbf{r}_N$ and the M previously examined code sequences of varying lengths), the decoder chooses to extend the most likely path (to within the quantization of the threshold spacing, Δ). The use of this metric results in a very "efficient" search for the correct path, and acts to mitigate somewhat the sequential decoder computational problem. Note that the M partial code words $\{\mathbf{t}_{k_1}, \mathbf{t}_{k_2}, \ldots, \mathbf{t}_{k_M}\}$ in general will have different lengths and the metric used will account for these differences. For each path $\mathbf{t}_{k_i}$ the metric (7-53) depends only on that path and the received symbols through the first $k_i + 1$ nodes. This is a result of the assumption that the decoder has no knowledge of the symbols in any extensions of that path. However, the use of this metric does not imply that the decoder will achieve maximum-likelihood decoding. The metric (7-53) is equivalent to a maximum-likelihood metric only when comparing paths with the same lengths. This metric is biased such that longer paths are favored over shorter paths to produce an efficient search. Because of this bias, a path that would have been selected by a maximum-likelihood decoder can fail to be extended by a sequential decoder. Fortunately, the impact of this nonoptimality is not severe.

7.2.3. Code Selection

Selection of codes for use with sequential decoding is not particularly critical as it is for the other techniques that have been considered. Any code can be decoded by any of the sequential decoding algorithms. In addition, decoder complexity is not strongly dependent on code constraint length. Thus, it is not necessary to optimize the code for a given value of v. Instead, one can increase v as much as necessary to achieve the desired level of performance. Since finding the optimum code (in the sense of the best weight structure) is computationally infeasible for large values of v, most researchers have been content to find codes with large values of d_f. The generators for a number of long codes suitable for sequential decoding are given in Appendix B in Table B-17. A long $R = 1/2$, systematic code found by Forney[78] and given in this table has been shown to have performance adequate for most applications. The resulting code generator is *nested*. That is, the generator may be shortened at any point to produce a good shorter code.

When it is desirable to minimize v, a nonsystematic code can be used. Several examples are also given in Table B-12. A very interesting nonsystematic code is the so-called "quick-look code" found by Massey and Costello.[79] The generators are related by

$$g_1(x) + g_2(x) = x \tag{7-54}$$

This allows recovery of the transmitted information sequence from the received sequence by simply adding $R^1(x)$ and $R^2(x)$, i.e.,

$$\begin{aligned} R^1(x) + R^2(x) &= I(x)g_1(x) + E^1(x) + I(x)g_2(x) + E^2(x) \\ &= xI(x) + E^1(x) + E^2(x) \end{aligned} \tag{7-55}$$

These codes eliminate one of the disadvantages of nonsystematic codes [that $I(x)$ cannot be reliably estimated without decoding], and the resulting error rate of recovered estimate of $I(x)$ is only twice the channel error rate (an "error amplification factor" of 2). However, one needs a larger value of v to achieve the same performance as a nonsystematic code without this feature.

In summary, an adequate list of suitable codes is already available. However, if for some reason an application requires an even longer code it would not be difficult to generate such a code.

7.2.4. Sequential Decoder Computational Problem

A major problem with sequential decoding is that the number of computations required in advancing one node deeper into the code tree is a

random variable. This characteristic strongly affects the complexity required to achieve a given level of performance. When there is little noise, the decoder is usually following the correct path requiring only one computation (i.e., one F, L, or B move) to advance one node deeper into the code tree. However, if the noise becomes severe, the metric along the correct path may decrease while the metric along an incorrect path may temporarily be larger. This causes the decoder to proceed along an incorrect path, and a large number of computations may be required before the decoder begins following the correct path. The number of computations required in getting through a period of severe noise is a rapidly increasing function of the number of times the threshold must be lowered. The most significant effect of the variability in the number of computations required in decoding a symbol is that a large memory is required to buffer the incoming data. Any finite buffer used with sequential decoding has a nonzero probability of overflowing, an event which must be considered in performance calculations.

This computational problem has been studied by researchers for a number of years. One of the more important results is the lower bound to the *distribution of computation* found by Jacobs and Berlekamp.[80] Their results are applicable to any sequential decoding algorithm. Such an algorithm is required to have only two properties:

1. Branches are examined sequentially so that, at any node, the decoder's choice among previously unexplored branches does not depend on received branches deeper in the tree.
2. the decoder performs at least one computation at each node of every path examined.

Rather than show the derivation of this bound, we shall discuss the simple example given by Jacobs and Berlekamp which illustrates the critical nature of the problem. Then the general result will be stated without proof.

Consider a binary erasure channel with erasure probability, δ (symbols are either erased or received correctly). Assume that an $R = m/n$ code is used, i.e., a code with 2^m branches per node and n symbols per branch. The probability of a burst of N erasures occurring in the first N symbols is δ^N. Assume that N is a multiple of n for simplicity. Note that after receiving erasures over these first N symbols, the decoder has absolutely no information with which to guide its selection of paths. The total number of paths (that are N symbols long) diverging from the origin is 2^{RN}. The decoder cannot use information beyond the first N symbols to order its initial search of these 2^{RN} paths. Thus, with probability 1/2, at least 1/2 of these paths must be examined before locating the correct path (with at least one computation per path). This means that with probability 1/2, at least

$$L = 2^{RN-1}$$

total computations will be required before the correct path can be followed.

Now let C denote the number of computations required to decode the first M information symbols, and choose

$$N = \frac{1}{R}\log_2(2L)$$

Note that the only values of N allowed are integer multiples of n, and it is necessary that $NR < M$. Then if the first N symbols are erasures, L or more computations occur with probability at least 1/2 giving

$$\begin{aligned}\Pr[C > L] &> \tfrac{1}{2}\delta^N \\ &= \tfrac{1}{2}2^{(NR\log_2\delta)/R} \\ &= \tfrac{1}{2}(2L)^{(\log_2\delta)/R} \\ &= aL^{-\alpha}\end{aligned}$$

where the parameters $\alpha = -(\log_2\delta)/R$ and $a = 2^{-(1+\alpha)}$ have been defined. A distribution of the form $aL^{-\alpha}$ is called Pareto with exponent α. Unless α is very large, this distribution decreases slowly with L creating the necessity for a large buffer.

This example demonstrates several properties of sequential decoding algorithms. The parameters α and a in this bound depend only on the *channel* and the *code rate*. Obviously then, the computational problem is reduced by increasing the signal-to-noise ratio. Note that the Pareto distribution arises because, even with random noise, there is a sufficiently high probability that long bursts will occur with the result that they require a very large number of computations in order to find the correct path. The probability of a noise burst of length N decreases exponentially with N, while the number of computations required increases exponentially with N. The result of these two opposing effects is the $L^{-\alpha}$ form of the Pareto distribution. This example also demonstrates that sequential decoders are extremely sensitive to burst error channels, i.e., any channel for which the probability of a burst error of length N does not decrease exponentially with N.

Sequential decoder computational behavior can be accurately described in terms of the "Gallager function," $E_0(\rho)$, given by (1-64) and the exponential bound parameter R_0 which were both discussed in Section 1.4. Jacobs and Berlekamp[80] showed that, for a discrete memoryless channel, the distribution of computation with any sequential decoding algorithm is lower bounded by

$$\Pr[C \geq L] > L^{-\rho}[1 - o(L)] \tag{7-56}$$

where the Pareto exponent ρ for a code of rate R on this channel is given

by the implicit solution of

$$R = \frac{E_0(\rho)}{\rho} \tag{7-57}$$

In this result $o(L) \sim 1/(\log_2 L)^{1/2}$. The result is valid for $0 < \rho < \infty$ and $0 < R < C$, where C is the channel capacity. The interesting point that this bound shows is that no sequential decoding algorithm can have a distribution of computation that decreases more rapidly with L than a function of the form $L^{-\rho}$. The resulting exponent in this bound is a function only of the code rate R and the channel, and it is easily calculated through (7-57). The exponent determined in this manner provides a tighter bound than the exponent found in the preceding example. Experimental and theoretical results, particularly the upper bound to the distribution of computation found by Savage,[81] support the conclusion that the actual distribution is Pareto and that the actual exponent can be found from (7-57). Knowledge of the operating Pareto exponent ρ allows one to accurately estimate the buffer size required to achieve a given performance level (buffer overflow probability). Thus, (7-56) and (7-57) are important in the decoder design process.

Note that the exponential bound parameter R_0 as given by

$$R_0 = E_0(1) \tag{7-58}$$

is the code rate at which $\rho = 1$ is the solution of (7-57). This rate is called the *computational cutoff rate* and is sometimes denoted by R_{comp}. This is a *practical limit* on the highest rate at which a sequential decoder can operate since a Pareto distribution with $\rho = 1$ is a distribution with an infinite mean. This indicates that any sequential decoder operating at $R > R_0$ will have a severe computational problem with frequent buffer overflows. The minimum feasible value of E_b/N_0 at which a sequential decoder can operate is usually predicted through calculating R_0. Since codes are usually chosen so that undetected error rates are extremely small under typical operating conditions, the computational problem has the dominant effect on system performance. Thus, R_0 is an important performance parameter.

7.2.4.1. Behavior with Coherent PSK. Consider the case of binary antipodal signaling over an additive Gaussian channel. The practical limit on sequential decoder performance for a code of rate R is the value of E_b/N_0 required to achieve $R_0 = R$. This parameter was evaluated in Section 1.4 for the case of PSK signaling with coherent detection. Required values of E_b/N_0 as a function of R are shown in Fig. 1-12. Demodulator quantizations of $Q = 2$, 8, and ∞ are shown. Operation at values of E_b/N_0 below those shown will result in an exceedingly severe computational load on the decoder. Note that with $R = 1/2$ codes, the required E_b/N_0 is 4.6 dB with hard decisions and 2.6 dB with eight-level quantization. The latter number is substantially better than can be achieved with Viterbi decoding of short codes.

In addition to the practical limit on E_b/N_0 one can also find the Pareto exponent at any E_b/N_0 above this limit. Consider the case of a binary symmetric channel with transition probability, p. Then the Gallager function can be written

$$E_0(\rho) = \rho - \log_2\{[(1-p)^{1/(1+\rho)} + p^{1/(1+\rho)}]^{(1+\rho)}\} \qquad (7\text{-}59)$$

The Pareto exponent, ρ, is given by the implicit solution of

$$R = 1 - \frac{1}{\rho}\log_2\{[(1-p)^{1/(1+\rho)} + p^{1/(1+\rho)}]^{(1+\rho)}\} \qquad (7\text{-}60)$$

For PSK signaling $p = Q[(2E_s/N_0)^{1/2}]$, and the resulting exponent for several code rates as a function of E_b/N_0 is shown in Fig. 7-14. In addition, similar results are shown for binary PSK signaling with eight-level demodulator decisions. Typical operating points in each case are the ranges of E_b/N_0 which produce $1 < \rho < 2$. Pareto exponents as predicted from Fig. 7-14 agree closely with those measured through simulation.

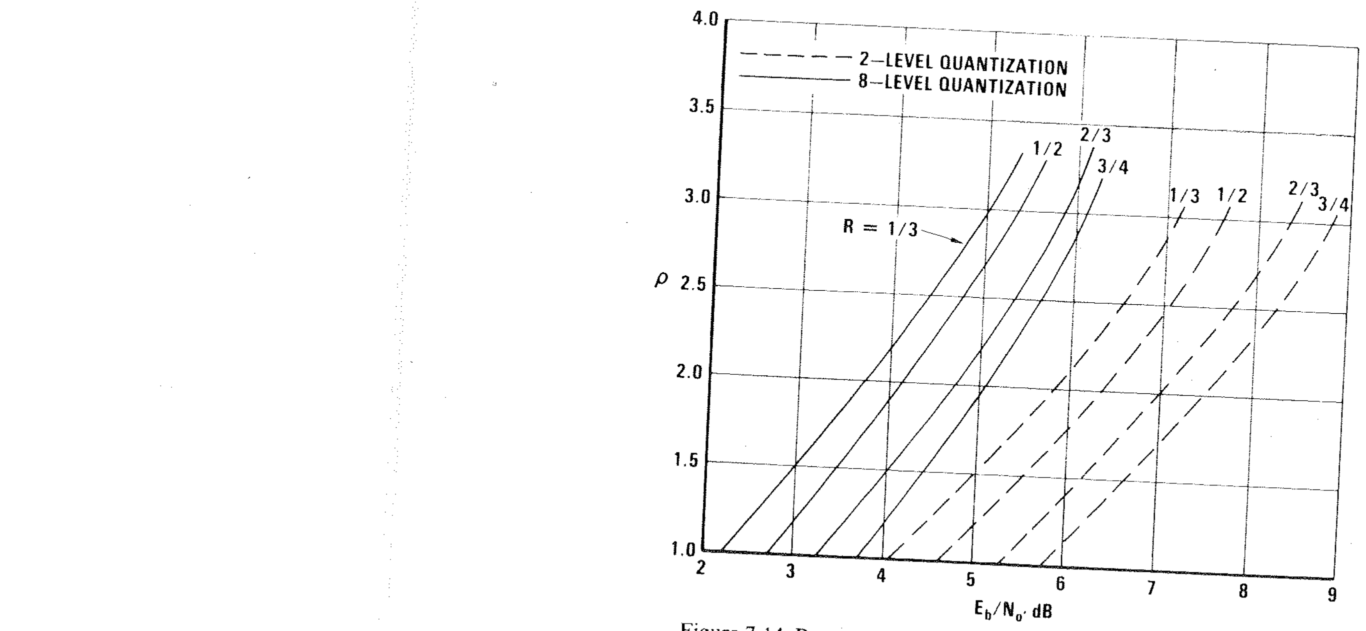

Figure 7-14. Pareto exponent as a function of E_b/N_0 and code rate.

7.2.4.2. Behavior with M-ary Orthogonal Signaling and Noncoherent Detection. We observed in Chapter 6 that a noncoherent system employing Viterbi decoding is most efficient when M-ary orthogonal signaling is used. Jordan has observed that this is also true in a system employing sequential decoding.[82] This is shown through the behavior of the R_0 parameter. We shall briefly summarize his results.

The demodulator for M-ary orthogonal signaling with noncoherent detection consists of a bank of M matched filters and envelope detectors. When these are sampled, the probability density function for the filter with signal present, $p_{s+n}(y)$, is given by (1-10).

For the case of signal absent, the probability density function, $p_n(y)$, is given by (1-11).

Jordan showed that with $Q = \infty$, the R_0 parameter can be expressed as

$$R_0 = \log_2\left\{M\left[1 + (M-1)\left(\int_{-\infty}^{\infty}[p_{s+n}(y)\,p_n(y)]^{1/2}\,dy\right)^2\right]^{-1}\right\} \qquad (7\text{-}61)$$

Then R_0 is found by substituting (1-10) and (1-11) into (7-61) and performing the integration numerically. This has been done for several values of M, and the E_b/N_0 required to achieve $R_0 = R$ as a function of R is shown in Fig. 7-15. The code rate R is expressed in information bits per M-ary symbol.

Several aspects of this figure are deserving of comment. First, we note that, as expected, higher-level signaling alphabets are significantly more efficient in terms of E_b/N_0 than binary signaling. For example, the advantage of 16-ary signaling relative to binary is over 4 dB. Second, we note that for each value of M, there exists an optimum code rate which is typically slightly greater than $\log_2 M/2$. With binary signaling $R = 1/2$ codes are optimum. This is consistent with the results found in Chapter 6 with Viterbi decoding. In comparison with those results, a $R = 1/2$ sequential decoder could perform about 1 dB better than a Viterbi decoder. For larger values of M, the simplest way to utilize M-ary signaling is to use one M-ary symbol per code branch. Then the code rate selected will determine the number of branches emanating from each node in the tree (e.g., $R = 1$ gives a binary tree, $R = 2$ gives a quaternary tree, etc.). Using this approach, a binary tree ($R = 1$) is optimum for $M = 4$. At $M = 8$, the quaternary tree ($R = 2$) is most efficient, but one could use a binary tree at a cost of only 0.4 dB. Similarly, at $M = 16$, the quaternary tree is most efficient. This last system was implemented by Jordan,[82] and his measured performance results compare closely with the required E_b/N_0 as predicted by Fig. 7-15 after adjusting for nonideal filtering and quantization. Finally, this figure also implies that not only is there an optimum code rate to use with sequential decoding

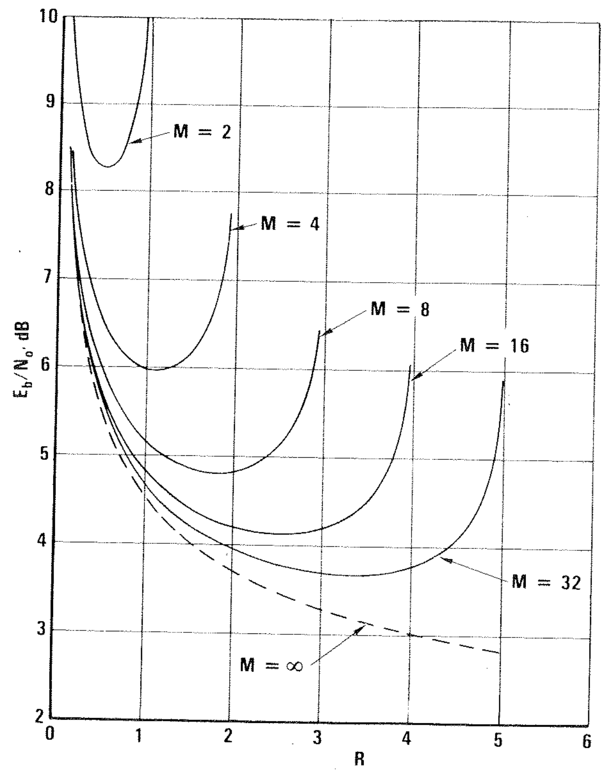

Figure 7-15. E_b/N_0 required to achieve $R = R_0$ as a function R for M-ary orthogonal signaling with noncoherent detection.

but there is also an optimum code rate to use in a system employing Viterbi decoding. Remember that the M-ary codes presented in the previous chapter utilized only $R = 1/2$ or $R = 1$ codes. The dual-3 code presented was a $R = 1/2$, 8-ary code. Figure 7-15 predicts that $R = 1/2$ codes are about 0.8 dB less efficient than $R = 1$ codes. The reason the $R = 1/2$ code was chosen is that the code constraint length is so short that the additional minimum distance is needed. Similarly, this figure also shows that $R = 1$ codes are near-optimum for 8-ary signaling. However, for larger alphabets ($M = 16$ or 32), one may wish to use $R = 2$ codes.

7.2.4.3. Simulation Results. The decoder computational problem has a large influence on the decoder design since the decoder computational burden must be estimated. This is typically done by estimating the decoder cumulative distribution of computations and the average number of com-

putations required per decoded branch. These quantities are influenced by code rate, E_b/N_0, and various decoder parameters such as metric assignments and threshold spacing.

First consider the influence of the metric assignment on the average number of computations. The form of the branch metric is given by (7-52). For the BSC the quantity added to the path metric when the received symbol agrees with the hypothesized channel symbol is

$$m_0 = \log_2 2(1 - p) - B \tag{7-62}$$

and the quantity added when the received symbol disagrees with the hypothesized channel bit symbol is

$$m_1 = \log_2 2p - B \tag{7-63}$$

The value of the bias should be set to $B = R$ to produce an efficient search. The general effect of using a bias term larger than R is to cause the decoder to enter a search more quickly as the result of a small amount of noise. This is desirable from an undetected error rate viewpoint since it increases the chance that the correct path will be examined (thereby providing an undetected error rate closer to that of a maximum-likelihood decoder). However, it is undesirable from a computational viewpoint because of the increased computational burden. Making B too small can also increase the computational burden because errors may not be recognized quickly enough.

To find the proper metric ratio, MR $= m_0/m_1$ for the BSC at $R = R_0$, (1-67) is solved for the corresponding value of p. Then by using this value of p and by setting $B = R$ in (7-62) and (7-63), the corresponding metric ratio is obtained. The resulting values of p and MR for $R = 1/2$, $2/3$, and $3/4$ are shown in Table 7-3. (Note that the optimum metric values for a $R = 1/2$ code at $R = R_0$ are approximately $+0.5$ for each symbol agreement and -4.5 for each symbol disagreement, which were the values used in the example in Section 7.2.1.)

Simulation results showing the average number of computations per decoded branch, $\bar{C}$, as a function of E_b/N_0 for a $R = 1/2$, $\nu = 15$ systematic

Table 7-3. BSC Transition Probability p and Optimum MR at $R = R_0$

R	p	MR
1/2	0.045	1/−9.15
2/3	0.017	1/−18.0
3/4	0.009	1/−27.7

code with several metric ratios are shown in Fig. 7-16. At $R = R_0\,(E_b/N_0 =$ 4.6 dB), the choice of MR $= 1/-9$ is far superior to the others. For higher values of E_b/N_0 this produces the minimum $\bar{C}$, but other choices such as MR $= 1/-11$ require only slightly more computations on the average. The latter choice may be preferable in this region due to an improved undetected error rate. [Note that MR $= 1/-11$ is optimum at $E_b/N_0 = 6.3$ dB as predicted by (7-62) and (7-63). The simulation results are consistent with this observation.]

The value of $\bar{C}$ is also a function of Δ, the threshold spacing. However, $\bar{C}$ exhibits a rather broad minimum with respect to Δ. For a metric ratio of $1/-n$ the value $\Delta = n + 1$ is nearly optimum and is also convenient to implement.

The distribution of computations is very important in estimating decoder buffer requirements. The tail of the cumulative distribution of computations per decoded branch has the form

$$\Pr[C \geq L] \approx AL^{-\rho}, \qquad L \gg 1 \tag{7-64}$$

where A is a constant usually of the order of 1 or 2, and ρ is the Pareto exponent given by (7-57). This exponent can be found directly from Fig. 7-14 for a coherent PSK system. The constant A depends on the specific algorithm and decoder parameters. Optimization of decoder parameters such as the metric assignments and the threshold spacing can reduce the constant, A. Improper metric assignments or demodulator AGC setting has also been known to degrade the Pareto exponent with soft decision decoding. Simulation results for a Fano algorithm implementation with a

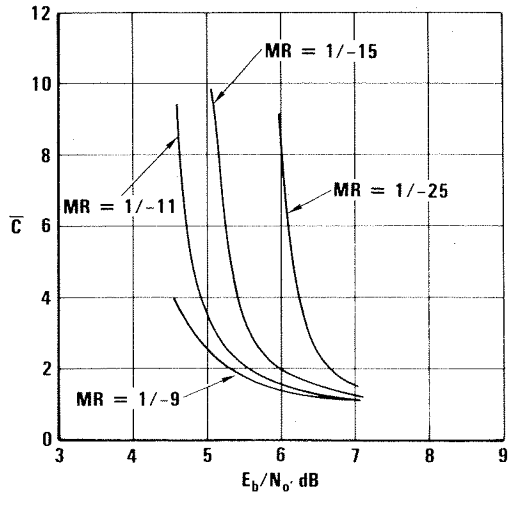

Figure 7-16. Average number of computations for a $R = 1/2$, $\nu = 15$ systematic code with hard decisions.

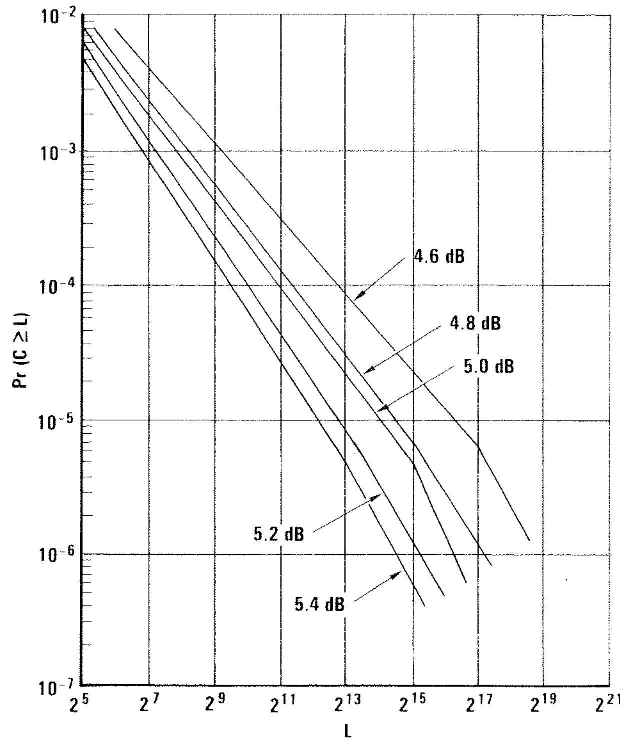

Figure 7-17. Distribution of computations for a $R = 1/2$, $\nu = 40$ systematic code with hard decisions.

$R = 1/2$, $\nu = 40$ code are shown in Fig. 7-17 (again hard decision quantization is assumed). This figure shows the measured $\Pr[C \geq L]$ when MR $= 1/-9$ and $\Delta = 10$ are used. The Pareto exponents given by these distributions correspond closely to those that would be predicted from (7-57). Thus, the tail of this distribution can be predicted by (7-64) with a constant A near unity.

7.2.5. Performance of Sequential Decoders

Exact analytical calculations of performance cannot be made for sequential decoders as is done with some other decoding techniques. However, performance can be estimated within a few tenths of a dB for most cases of interest by using certain approximations. Decoding errors can arise from two sources. The choice of code and metric ratio determines the undetected error rate, P_e. This is the bit error rate that would be observed if