# PERFORMANCE RESULTS FOR A HYBRID CODING SYSTEM

**L. B. HOFFMAN**
**Ames Research Center**
**NASA**
**Moffett Field, California 94035**

**Summary** Computer simulation studies of the hybrid pull-up bootstrap decoding algorithm hive -been conducted using a constraint length 24, nonsystematic, rate 1/2 convolutional code for the symmetric channel with both binary and 8-level quantized outputs. Computational performance was used to measure the effect of several decoder parameters and determine practical operating constraints. Results reveal that the track length may be reduced to 500 information bits with small degradation in performance. The optimum number of tracks per block was found to be in the range of 7 to 11. An effective technique was devised to efficiently allocate computational effort and identify reliably decoded data sections. Long simulations indicate that a practical bootstrap decoding configuration has a computational performance about 1.0 dB better than sequential decoding and an output bit error rate about 2.5 x$10^{-6}$ near the $R_{comp}$ point.

**Introduction** The basic coding dilemma is one of exponentially increasing decoding complexity as the theoretical capacity of a communications channel is approached. Hybrid coding is a cascade or concatenation of block and/or convolutional codes in an attempt to operate near capacity while maintaining a complexity less than that possible with either code type alone. This paper presents the results of a study of the hybrid bootstrap coding system of Jelinek.[1] This technique is similar to a simple case of the Falconer scheme, in that a parity relationship between a set of convolutionally encoded data tracks is used to aid in the decoding of those portions that are difficult. (An even parity is assumed throughout.) It differs from the Falconer scheme, which uses an algebraic relationship to derive directly the most difficult portions after a sufficient number of others are decoded, by making use of additional probabilistic information contained in the parity relationship. In so doing, each bit of data decoded helps to "bootstrap" those remaining.

After reviewing briefly the functioning of bootstrap decoding, this paper examines the computational effect of several decoder parameters and determines a practical range of operating values. Detailed performance behavior of such an optimized system is presented and compared to simple sequential decoding and Falconer decoding.

**Encoding** The encoding function is the same for all variations of bootstrap decoding described in this paper. (The decoders differ only in the manner in which they utilize information that is always available at the receiver.) Basically, m-I independent, convolutionally encoded “data tracks” are linked together into one “decoding block” by the addition of an m-th “parity track.” That is, each bit of the parity track is the modulo-two sum of the corresponding bits in the data tracks. Because of the linearity of convolutional codes, this parity track is also decodable and, as will be shown, may actually be generated by a convolutional encoder. The reader will note that this encoding function is identical to that required by the Falconer system for a simple parity check code.

Actual mechanization of the encoder depends upon several operational considerations. One method, which requires in-1 convolutional encoders, provides natural interleaving of the tracks. Data are routed to the encoders for coding and transmission in a “round robin” fashion, with the parity bit inserted in its turn by a modulo-two adder. Decoder synchronization for such a scheme will be difficult; synchronization and tail-forcing bits must be independent of data formating, possibly causing a small data buffering problem at the end of each block. Failure of the decoder to complete the decoding of a block results in the loss of a large amount of data, if not the entire block.

An alternative way of mechanizing the encoder requires one convolutional encoder and a storage register having the length of a track. The data are encoded and transmitted, one track at a time, while the parity track is formed in the storage register. The contents of the storage register are then encoded, transmitted, and reset following the last data track of each block. Although this scheme does not provide interleaving and causes an even larger buffering problem while the parity track is transmitted, it does offer several advantages. It is possible to let data formating correspond to individual data tracks. Code synchronization can be performed easily on a track basis, with block synchronization derived from identification bits embedded in the data tracks. In addition, a decoder failure will not necessarily result in the loss of a full block of data. Finally, since data formating, synchronization, and tail-forcing can be related, the rate loss for these functions can be reduced.

**Rudimentary Bootstrap Decoding** Bootstrap decoding is applicable to all symmetrical binary input channels. For the purposes of this paper,, a simplified description of the “rudimentary” algorithm for the binary symmetric channel (BSC) is given following the outline used by Jelinek.[1]

After the encoded data have been received and are synchronized, the bits of a block are grouped into m tracks, and an additional track, the “channel state stream,” is formed by the decoder. Each channel state stream bit is the modulo-two sum of corresponding bits in the parity and data tracks. The channel state stream differs from the parity track because it includes the parity track and is formed after the transmitted sequence is corrupted by noise. Therefore, a “zero” in this track indicates that an even number of errors was received at a given position, and a “one” indicates an odd number.

The probabilities that k bits which are independently transmitted through a BSC of crossover probability p will be received with an even or odd number of errors are given by

$$q_k(0) = [1 + (1 - 2p)^k]/2$$
$$q_k(1) = [1 - (1 - 2p)^k]/2$$

The information is used to form an augmented transition probability matrix $w_m(y,z/x)$ where y is the received bit and z is the channel state bit associated with y and formed over m tracks, given that x was transmitted. Thus:

$$\begin{aligned} w_m(0,0/0) &= w_m(1,0/1) = (1 - p)q_{m-1}(0) \\ w_m(0,1/0) &= w_m(1,1/1) = (1 - p)q_{m-1}(1) \\ w_m(1,0/0) &= w_m(0,0/1) = \quad p \quad q_{m-1}(1) \\ w_m(1,1/0) &= w_m(0,1/1) = \quad p \quad q_{m-1}(0) \end{aligned}$$

It is natural to use these augmented transition probabilities in forming the bit likelihood function for sequential decoding. The function is

$$\lambda_m \triangleq \log_2 [w_m(y,z/x)/w_m(y,z)] - R$$

where

$$w_m(y,z) = [w_m(y,z/0) + w_m(y,z/1)]/2 = [q_m(z)]/2$$

and R is the bias factor.

From this starting point, the development of the rudimentary bootstrap decoding algorithm follows directly. The first of the m tracks is sequentially decoded using the channel state stream and likelihood values defined above. If, after a preassigned amount of effort, decoding of this track is not completed, restart values are saved. This step is repeated on successive tracks, looping back to the first track if necessary, until decoding of one is completed. At this time, the received sequence for the completely decoded track is replaced by the newly estimated sequence, and the channel state stream is recomputed. If the decoding was error free, then the new channel state stream values represent an even or odd number of errors in the m-1 remaining tracks, as before. The entire process is repeated, excluding the decoded track, now using likelihood values for m+1 tracks. When a second track is completed, its received sequence is replaced, and the channel state stream is again updated.

The pattern is now obvious, and the process is repeated until all tracks have been decoded or the total work exceeds a maximum amount. It would be possible to derive the last remaining track, on the basis of the parity relationship, when rn- I tracks have been decoded. Indeed, this is the principle of Falconer decoding; but it is actually simpler to decode this track, too, since the decoding requires exactly one computation per bit. This fact, and the general effect of using the channel state stream, may be seen in the sample likelihood table shown in figure 1. When many tracks are undecoded, the channel state bit gives little additional information about the probability of error in a single received bit. Therefore, for large k, the likelihood values for bootstrap decoding approach the usual values for sequential decoding, depending mainly upon agreement or disagreement between the received bit and the hypothesis. At the other extreme, for small k, the channel state bit has a large influence. For example, if two tracks remain undecoded (k = 2) and the channel state bit is “one,” neither hypothesis is reliable because the probability is 0.5 that the received bit is in error. On the other hand, great reliance is placed on the correctness of the received bit when the channel state bit is “zero” since the probability of a double error is small. When k = 1, the knowledge that the received bit is in error for a channel state bit “one” and correct for a “zero” is reflected in the table by a ∞ likelihood value for the impossible hypothesis and 1.0 for the correct hypothesis.

**Pull-Up Algorithm** The primary worth of the rudimentary algorithm is the description of the bootstrapping process and simplification of its analysis. Practical use of the rudimentary algorithm is probably limited because one rather simple modification substantially increases the power of the decoder. In the modified algorithm, called the “pull-up” algorithm, the decoder does not wait until a track is decoded completely before updating the channel state stream. It operates instead on a single track until the track is completed or a difficult-to-decode section is sensed, at which time decoding is stopped. The completed track is handled as in the rudimentary algorithm. Before proceeding with the next track after a track is terminated, however, the decoder declares that portion which it deems reliable to be “definitely decoded.” In doing so, it updates the channel state stream and prepares restart values so that the next decoding attempt on the track will begin immediately after the definitely decoded section.

Since it is possible to have all tracks in varying stages of completion, to obtain the most effective use of the channel state stream it is necessary to indicate how many tracks remain undecoded at a given node. This is done with a vector, KLEFT, the length of a track, which the decoder references to determine the likelihood values to use at a given node. At the outset, all KLEFT values are set to m, the number of tracks in a block, and are adjusted accordingly as individual tracks are “pulled up.” Note that it is necessary each time to start decoding from the “origin” because the state stream may change from the time the decoder terminates a track to the time the decoder restarts it.

**Computer Simulations** Many variables affect the performance and practicality of a system as complex as bootstrap decoding. Unfortunately, analysis can give only bounds on performance for simplified and idealized conditions. Therefore, simulations have been performed to determine the gross effect of a number of parameters for the pull-up version and to obtain performance figures for a quasi-optimized system that could be considered for possible deep space application.

The simulation program was written in FORTRAN for a 24-bit, 1.75 $\mu$s/cycle computer with in-line assembly language used to optimize the critical loops. The convolutional code was restricted to the rate 1/2, constraint length 24 complementary code (taps 51202215 and 66575563) found by Bahl and Jelinek.[3] This code was selected because it could be simulated within a single computer word and is sufficiently powerful (free distance 24, minimum distance 10) that decoder errors do not limit the system. The Fano algorithm was used for sequential decoding with a simulation speed in excess of 3000 computations per second. All simulations were run with the bias factor R = 0.5 and the threshold spacing = 3.0. One-dimensional parameter studies of this system using the BSC concern track length, tracks per block, stopping rule, and reliability criterion. These tests were run at low signal energy per information bit per noise power spectral density ($E_b/N_o$) values so that the effects of the parameters could be observed near threshold-of-operation conditions.

**Track Length** It is possible (perhaps desirable from a theoretical point of view) that the track length be very long for the pull-up algorithm. Other practical considerations, such as synchronization, formating, and buffering, require that the track length be reasonably short. Figure 2 shows the effect of track length on computation performance, with the number of tracks fixed at 7. All tracks for all simulations are terminated by a one-constraint-length tail that is included in the rate loss for the code. The value of $E_b/N_o$ was fixed at 3.43 dB, and for direct comparison, the computation distributions per block were normalized per information bit before being plotted. Average computations are shown in the legend. It can be seen that the computation performance is degraded for a track length of 300 information bits, but that little improvement is actually obtained for a length beyond 500. The track length was fixed at 500 information bits for all other simulations.

**Tracks per Block** The rate loss of bootstrap decoding, determined by the number of tracks per block, m, is significant for small m but it decreases rapidly and then changes relatively little as m becomes large. This fact, and the fact that the effect of the channel state stream is predominant when the number of tracks is small, suggests that an optimum value for m can be found. In addition, the value of m has a direct influence on formating, encoder complexity, and decoder buffering. Simulations were conducted to determine the effect of m on the computation distributions. The track length was fixed at 500 information bits (plus the one-constraint-length tail), and all simulations were carried out for a value of $E_b/N_o$ held constant at 3.43 dB. Figure 3 is a summary of the results of these simulations, with distributions of normalized computations per information bit plotted for selected values of m and a more complete table of average computations per information bit given in the legend. The irregular variation in computations between values of rn is probably due to small sample size ($3.675x10^6$ bits per value of m), but a broad minimum is indicated between m = 7 and 11. Values of m in this range would be practical for operational use. The number of tracks per block was fixed at 7 for all other simulations.

**Stopping Rule** An effective stopping rule must be devised in order to obtain the maximum efficiency of pull-up bootstrap decoding. The sequential decoder should be allowed to operate as far as it can go easily. Unnecessary time is wasted in restarting when a track is stopped too soon, or in computing, when it is not stopped soon enough. In addition, the stopping rule should provide information about the reliability of the path on which the decoder is operating. Several rules based upon limiting the number of computations per track were devised and tested, but none proved very useful because of the large variation in the number of computations for each track. When the computations limit is set low and increased when no progress has been made on any track, many decoding attempts are required to complete each block. Setting the initial limit high to reduce the number of attempts caused long unnecessary searches. In addition, computations alone do not provide reliability information.

The final and most effective rule devised is based solely on observation of the path likelihood value. Since the likelihood of the correct path tends to increase with depth in the code tree, the rule allows the decoder to operate as long as a drop in the value of the likelihood does not exceed a specified value, D. Mechanization of this rule also gives the needed reliability information. The decoder keeps track of the maximum likelihood value, $L_{max}$, of any path visited. Operation is stopped if the decoder attempts to lower the threshold more than D below $L_{max}$. At this time, the decoder is pointing to a node before the $L_{max}$ node which has a path likelihood approximately D below it. The probability that this node is on the correct path increases with increasing values of D. The definitely decoded section is declared to extend from the starting point up to LBACK nodes from the stopping point, where LBACK is another variable in the stopping rule.

In order to sense stagnation in the decoding process, it is necessary to count the times the definitely decoded section is not increased by NPULL nodes for a single decoding attempt. For all simulations, NPULL was set to 15. The counter, KROUND, is initially set to 0 and reset each time that decoding results in more than NPULL definitely decoded nodes. If the KROUND count becomes equal to the number of undecoded tracks, thus indicating that no progress can be made on any track, the value of D is increased and KROUND reset. At this time, the channel state stream is recomputed and decoding is begun from the first node of each uncompleted track. This procedure allows for correction of possible errors included in definitely decoded sections of the incomplete tracks which may be causing the decoding difficulty. The value of D is reset to its initial value each time a track is completely decoded.

Figures 4 and 5 show the results of simulations for the above scheme. All simulations are for 500 information bits per track, 7 tracks per block, and $E_b/N_o$ = 3.43 dB. For these simulations, D is determined by multiplying the indicated stop factor by the “disagree, 0 state bit” likelihood value for the number of existing uncompleted tracks. Figure 4 shows the effect of several stop factor sequences with LBACK = 50, and figure 5 shows the effect of LBACK using only the 4, 5, 6, 7 stop factor sequence. It can be seen that an initial stop factor of 3 or 4 is optimum with an increase of I each time stagnation occurs. For these values the stopping point does not usually contain errors, and LBACK may be small.

**Performance of Optimum System** Figure 6 shows the performance of a pull-up bootstrap decoder for the BSC. System parameters, chosen near the optimum values determined in the previous simulations, were held fixed over the $E_b/N_o$ range. Although no further attempt was made to optimize the system, these curves provide a good measure for comparison with other systems. The Pareto slope, α, is plotted as a function of $E_b/N_o$ in figure 7. The $R_{comp}$ point is interpolated to be 3.1 dB. During these simulations 62 decoder errors were observed for the 3.43 dB case. The resulting output bit error rate was about 5 x $10^6$.

It is worthwhile to note here that the power of the code and stopping rule worked very effectively in eliminating decoder errors. Numerous errors were inserted in partially completed tracks but were removed when the tracks were eventually restarted. The 62 errors occurred in one block; 31 were decoded into the second track to be completed and the other 31 were forced by parity into the last track. (Weaker codes have been observed to permit more frequent errors, which were also duplicated in a second track with no significant effect on the computation performance.)

Figure 8 shows pertinent information about decoder operation for one block of the $E_b/N_o$ = 3.43 dB sample. This block was selected because it shows the decoder trying to commit errors (step 7), a change in stopping rule (step 18), the effect of pull-up, and the general reduction in computations per track as the quantity of definitely decoded data is increased (when there are no errors). The step number is KTRY; JNOW is the track being operated on; ITCT is the number of computations for the step; IT is the stopping threshold value; ITMX is the maximum threshold value; DFAC is the stopping rule likelihood drop factor; NSTART is the starting node; N is the stopping node; NMAX is the maximum node depth; KLEFT is the number of uncompleted tracks after the decode step; and KROUND is the number of steps since pull-up.

Figure 9 is a plot of the probability that the total number of decoder steps per block will exceed a given number for the optimum system with Eb/No as a parameter. Note that these curves exhibit a Pareto-type distribution with a sharp change in slope near the Rcomp point of the system.

**Comparison With Other Systems** It is interesting to compare bootstrap decoding with two other decoding techniques because of their similarities. The first is simple sequential decoding. To provide a means for direct comparison, simulations were performed for the same $E_b/N_o$ values as were used for the bootstrap decoder. The same Fano algorithm, track size, and rate 1/2 convolutional code were used. The results are shown in the normalized computations curves of figure 10 with the Pareto slope values plotted in figure 7. $R_{comp}$ is at approximately 4.6 dB. Bootstrap decoding has a gain of about 1.5 dB over simple sequential decoding.

In order to determine the exact effect of the channel state stream, the pull-up decoder was modified to use standard likelihood values when k ranges from 2 to 7 so the channel state stream is useless, except to pull up the track which is farthest behind the others. Consequently, the algorithm actually behaves like the Falconer algorithm for a 7-bit parity check code, with the exception that the decoder is restarted from the first undecoded node at each decoding attempt. The computation results of these simulations are shown in figure 11 with the Pareto slope values plotted in figure 7. This algorithm has an $R_{comp}$ of about 4.1 dB which is only 0.5 dB better than simple sequential decoding. The use of the channel state stream therefore yields a rather inexpensive 1.0 dB gain.

**Extension to Quantized Channel** Bootstrap decoding would be of little use if it were applicable only to binary output channels since nearly 2 dB can be gained for simple sequential decoding if the output is quantized to eight levels. Jelinek has provided such an extension for the bootstrap decoder.[1] Unfortunately, to make full use of the information provided by the quantized symbols, a large amount of time is required to compute channel state values, which are no longer binary. Excessive computing time, coupled with the large likelihood tables required (15,280 entries for 8-level quantization and 7 tracks), probably makes such a scheme impractical.

Fortunately, there is a compromise - to use the quantized values of the track symbols and maintain only a binary channel state stream. If the receiver outputs are broken into sign and quality bits, u and v, then the channel state values, z, are modulo-two sums of u, as before. Then

$$\lambda_m \triangleq \log_2 [w_m(u,v,z/x)/w_m(u,v,z)] - R$$

where

$$w_m(u,v,z) = [w_m(u,v,z/0) + w_m(u,v,z/1)]/2$$

and

$$\begin{aligned}
w_m(0,v,0/0) &= w_m(1,v,0,1) = w(0,v/0)q_{m-1}(0)\\
w_m(0,v,1/0) &= w_m(1,v,1/1) = w(0,v/0)q_{m-1}(1)\\
w_m(1,v,0/0) &= w_m(0,v,0/1) = w(1,v/0)q_{m-1}(1)\\
w_m(1,v,1/0) &= w_m(0,v,1/1) = w(1,v/0)q_{m-1}(0)
\end{aligned}$$

$q_k(z)$ is defined as before, and

$$p = \sum_{v} w(1,v/0)$$

According to theoretical bounds derived by Jelinek,[1] full use of the 8-level channel gives an additional gain of about 1.7 dB over the BSC for rate 1/2 bootstrap decoding. Using a binary state stream for this channel causes a theoretical degradation of only 0.1 dB, which is a small price to pay since the channel state computation and likelihood look-up are direct and the table size is only four times larger than for the BSC.

The simulation program was modified for the quantized channel with binary state stream with no significant change in speed. Simulations were performed for eight levels of output with quantization spacing of 0.5 σ for all $E_b/N_o$. Tests were conducted which determined the optimum values for the stop factor sequence to be 2.0, 2.5, 3.0, 3.5 times the “strongest disagree, 0 state bit” likelihood with LBACK = 10, 7 tracks, 500 information bits per track, and $E_b/N_o$ = 1.91 dB. Extensive computer runs were made under these conditions for a range of $E_b/N_o$ values. The resulting computation performance curves are shown in figure 12. The observed Pareto slopes are plotted in figure 7 for comparison with the other simulations. The interpolated $R_{comp}$ point is at 1.7 dB, a gain of 1.4 dB over the BSC and 1.0 dB better than rate 1/2 sequential decoding using the octal channel. Figure 7 also shows an interesting thresholding effect for the codes plotted - the threshold is approached more sharply as code power increases. Over 27,000 blocks were run for the 1.91 dB case (near the threshold of operation) in order to look for any peculiar deviation in computations performance for low probabilities of C > T. The Pareto slope remained constant over the significant range. For this case, 190 bit errors were observed in 4 blocks for a probability of bit error less than $2.5 \times 10^{-6}$.

**Conclusions** Simulations have provided a great deal of experience with the bootstrap decoding algorithm. Although a number of questions remain unanswered (e.g., effects of channel memory and likelihood/channel mismatch), it is clear that this technique offers a gain of about 1.0 dB over that obtainable from sequential decoding alone. Bootstrap decoding has been shown to operate under the constraints imposed by digital communication systems, such as those typical of deep space. A bootstrap decoding system would be relatively complex, but appears suitable for low-to-moderate data rates where the value of 1.0 dB is worth the cost of implementation.

**Acknowledgement** The research for this paper was performed in conjunction with a study contract with Cornell University, Prof. F. Jelinek, principal investigator.

## Reference

1. F. Jefinek, and J. Cocke, "Bootstrap hybrid decoding for symmetrical binary input channels," Information and Control, March 1971.

2. D. D. Falconer, "A hybrid coding scheme for discrete memoryless channels," Bell System Technical Journal, vol. 48, pp. 691-728, March 1969.

3. L. R. Bahl, and F. Jefinek, "Rate 1/2 convolutional codes with complementary generators, to appear in the IEEE Transactions on Information Theory.

| RECEIVED BIT/ HYPOTHESIS | CHANNEL STATE BIT, z | NUMBER OF UNDECODED TRACKS, k | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 15 | 20 | 25 | 30 | 35 |
| AGREE | 0 | 1.000 | 0.986 | 0.972 | 0.959 | 0.947 | 0.936 | 0.926 | 0.917 | 0.909 | 0.902 | 0.879 | 0.870 | 0.866 | 0.865 | 0.864 |
| | 1 | $-\infty$ | .000 | .410 | .576 | .664 | .719 | .755 | .780 | .799 | .813 | .847 | .858 | .862 | .863 | .864 |
| DISAGREE | 0 | $-\infty$ | −5.690 | −4.717 | −4.618 | −3.797 | −3.525 | −3.318 | −3.156 | −3.027 | −2.924 | −2.638 | −2.535 | −2.496 | −2.482 | −2.477 |
| | 1 | 1.000 | .000 | −.576 | −.972 | −1.267 | −1.496 | −1.677 | −1.823 | −1.941 | −2.037 | −2.311 | −2.414 | −2.452 | −2.466 | −2.471 |

Fig. 1 – Likelihood values $\lambda_k$ for $p = 0.09$ and $R = 0.0$.

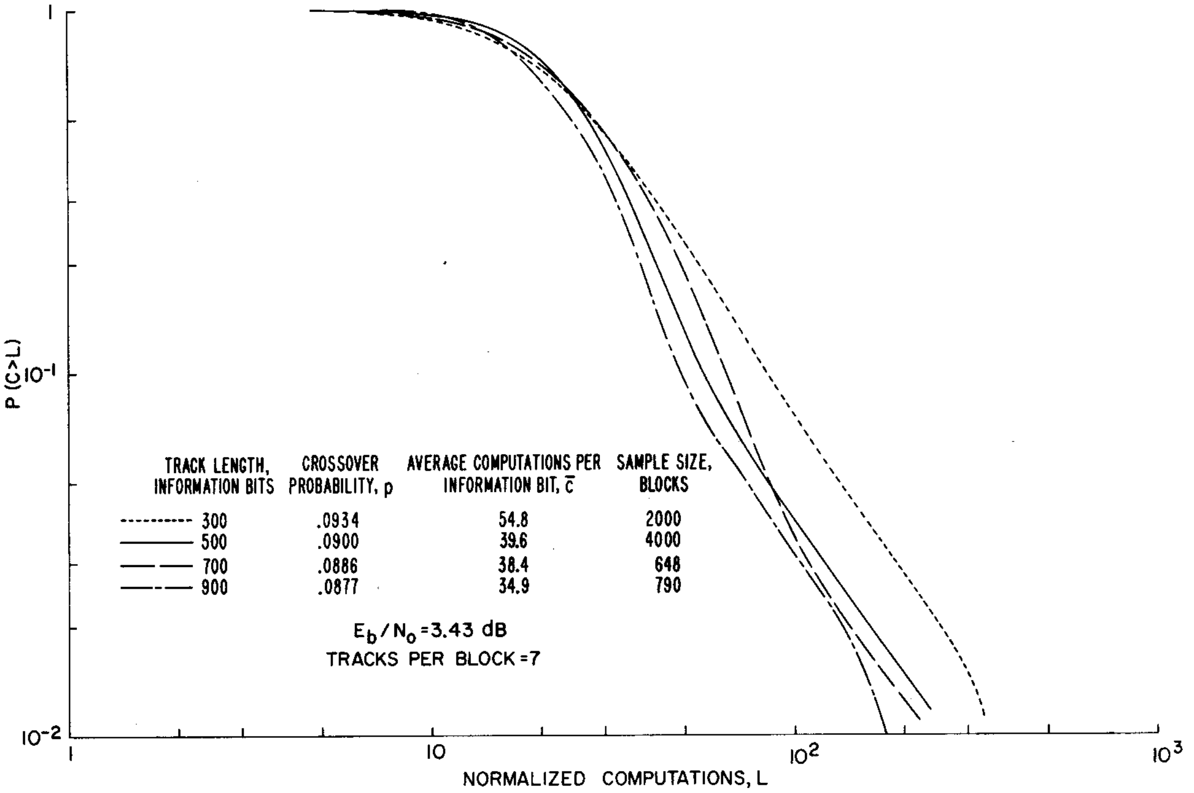
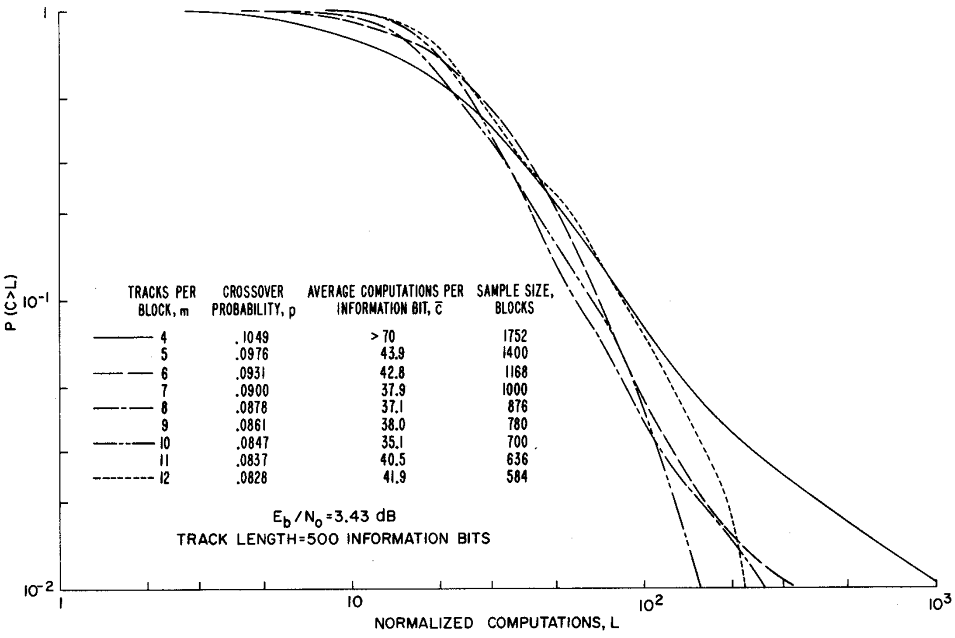


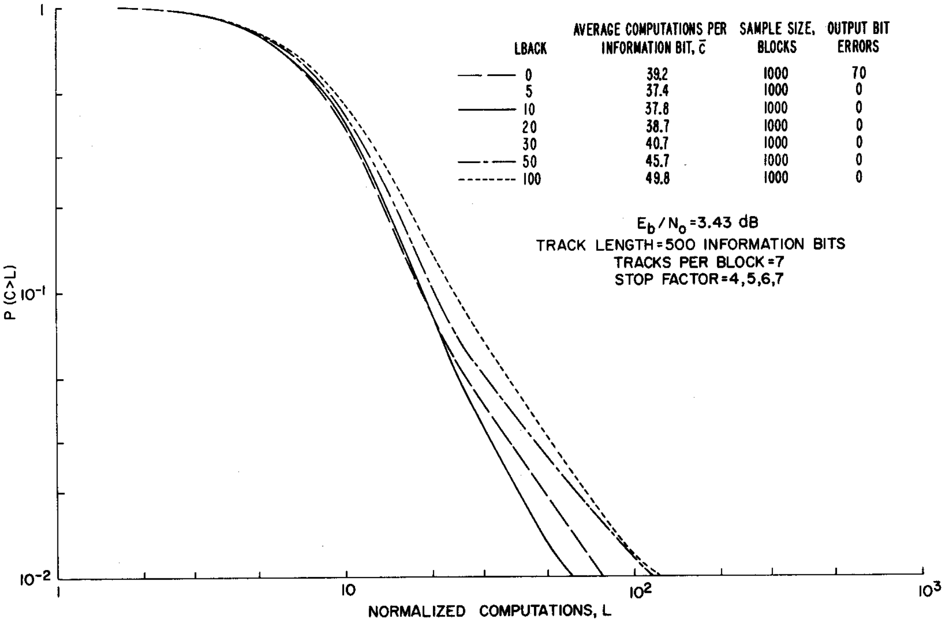
**Fig. 2 - Pull-up decoder computations performance as a function of track length.**



| TRACKS PER BLOCK, m | CROSSOVER PROBABILITY, p | AVERAGE COMPUTATIONS PER INFORMATION BIT, $\bar{c}$ | SAMPLE SIZE, BLOCKS |
|---|---|---|---|
| ——— 4 | .1049 | > 70 | 1752 |
| 5 | .0976 | 43.9 | 1400 |
| — — 6 | .0931 | 42.8 | 1168 |
| 7 | .0900 | 37.9 | 1000 |
| —·— 8 | .0878 | 37.1 | 876 |
| 9 | .0861 | 38.0 | 780 |
| —‑—‑ 10 | .0847 | 35.1 | 700 |
| 11 | .0837 | 40.5 | 636 |
| ------- 12 | .0828 | 41.9 | 584 |

$E_b / N_o$ =3.43 dB
TRACK LENGTH=500 INFORMATION BITS

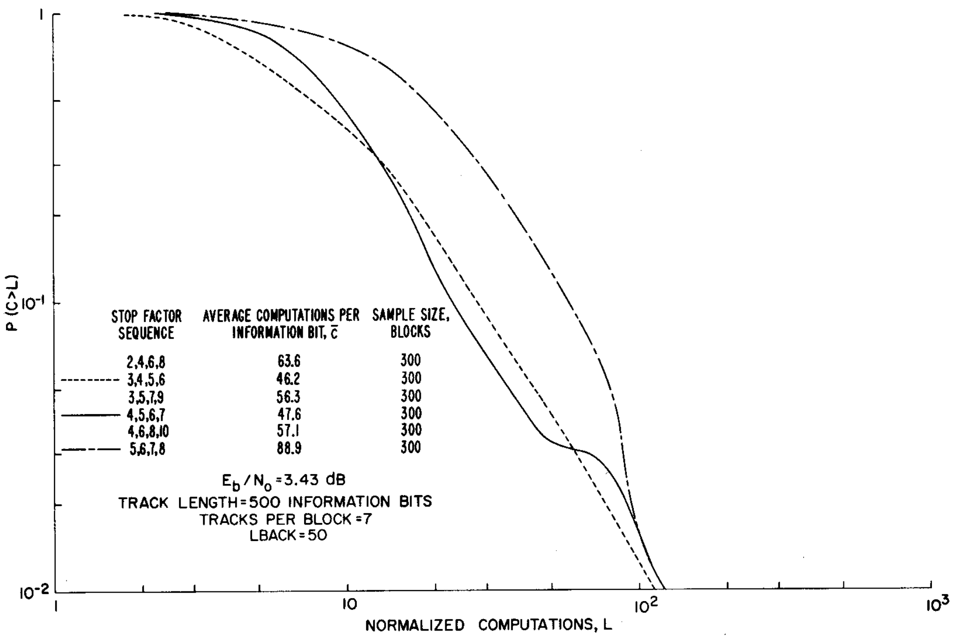
**Fig. 3 - Pull-up decoder computations performance as a function of tracks per block.**



| LBACK | AVERAGE COMPUTATIONS PER INFORMATION BIT, $\bar{c}$ | SAMPLE SIZE, BLOCKS | OUTPUT BIT ERRORS |
|---|---|---|---|
| — — 0 | 39.2 | 1000 | 70 |
| 5 | 37.4 | 1000 | 0 |
| ——— 10 | 37.8 | 1000 | 0 |
| 20 | 38.7 | 1000 | 0 |
| 30 | 40.7 | 1000 | 0 |
| —·— 50 | 45.7 | 1000 | 0 |
| ------- 100 | 49.8 | 1000 | 0 |

$E_b / N_o$ =3.43 dB
TRACK LENGTH=500 INFORMATION BITS
TRACKS PER BLOCK=7
STOP FACTOR=4,5,6,7

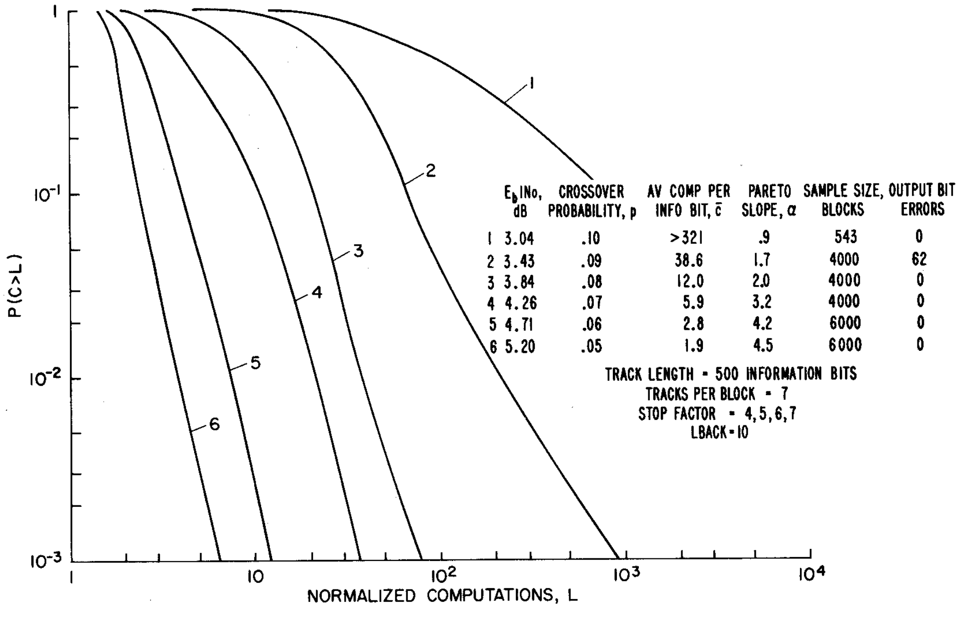
**Fig. 4 - Pull-up decoder computations performance as a function of stop factor.**



| | STOP FACTOR SEQUENCE | AVERAGE COMPUTATIONS PER INFORMATION BIT, $\bar{c}$ | SAMPLE SIZE, BLOCKS |
|---|---|---|---|
| | 2,4,6,8 | 63.6 | 300 |
| - - - - - | 3,4,5,6 | 46.2 | 300 |
| | 3,5,7,9 | 56.3 | 300 |
| ——— | 4,5,6,7 | 47.6 | 300 |
| | 4,6,8,10 | 57.1 | 300 |
| — — — | 5,6,7,8 | 88.9 | 300 |

$E_b/N_o = 3.43$ dB
TRACK LENGTH=500 INFORMATION BITS
TRACKS PER BLOCK=7
LBACK=50

**Fig. 5 - Pull-up decoder computations performance as a function of LBACK.**



| | $E_b/N_o$, dB | CROSSOVER PROBABILITY, p | AV COMP PER INFO BIT, $\bar{c}$ | PARETO SLOPE, $\alpha$ | SAMPLE SIZE, BLOCKS | OUTPUT BIT ERRORS |
|---|---|---|---|---|---|---|
| 1 | 3.04 | .10 | >321 | .9 | 543 | 0 |
| 2 | 3.43 | .09 | 38.6 | 1.7 | 4000 | 62 |
| 3 | 3.84 | .08 | 12.0 | 2.0 | 4000 | 0 |
| 4 | 4.26 | .07 | 5.9 | 3.2 | 4000 | 0 |
| 5 | 4.71 | .06 | 2.8 | 4.2 | 6000 | 0 |
| 6 | 5.20 | .05 | 1.9 | 4.5 | 6000 | 0 |

TRACK LENGTH - 500 INFORMATION BITS
TRACKS PER BLOCK - 7
STOP FACTOR - 4,5,6,7
LBACK-10

**Fig. 6 - Optimized pull-up decoder computations performance for the BSC as a function of $E_b/N_o$.**

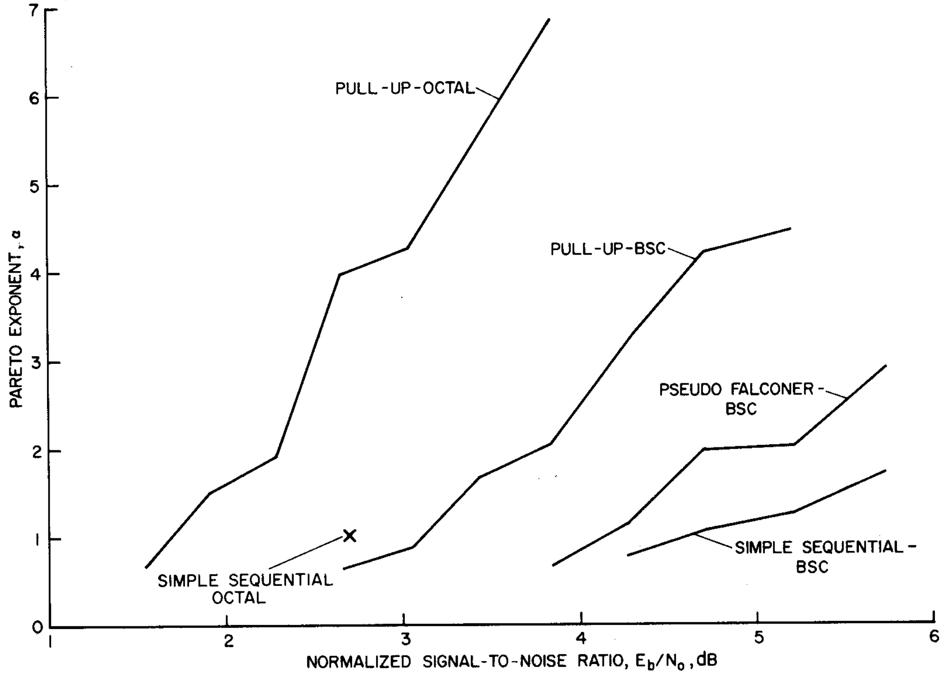
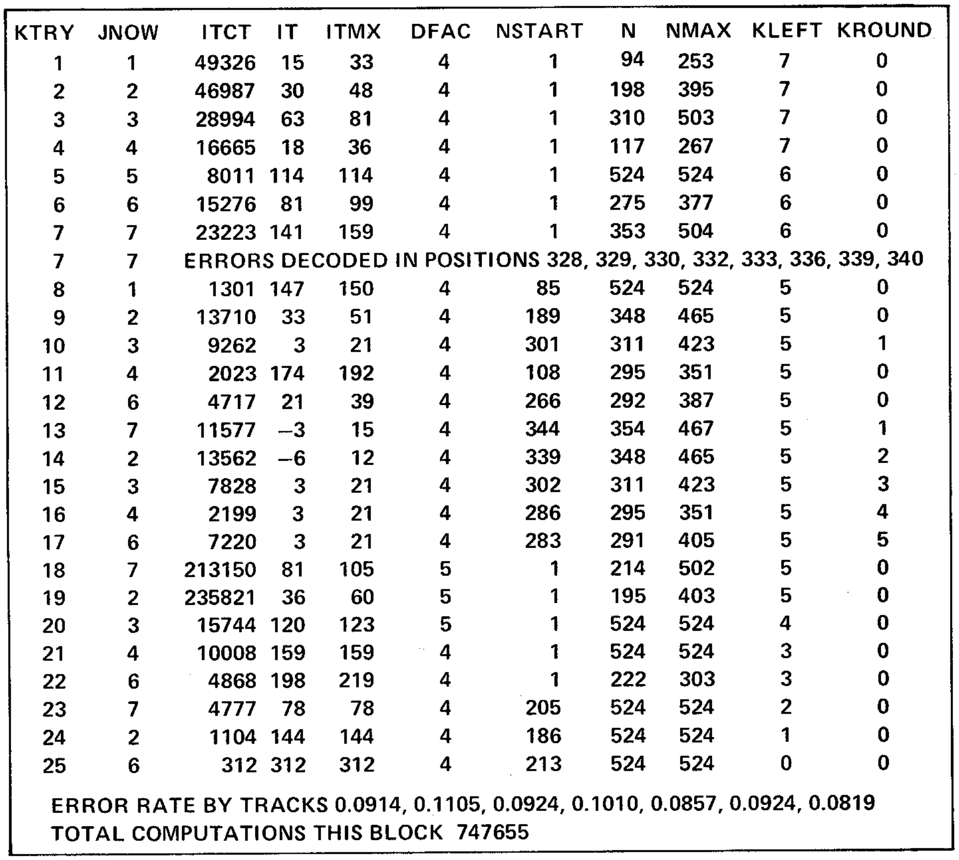


Fig. 7 - Pareto exponent vs. $E_b/N_o$ for several decoding techniques.



```
KTRY  JNOW    ITCT   IT  ITMX   DFAC   NSTART    N   NMAX  KLEFT  KROUND
  1     1    49326   15    33     4         1   94    253     7       0
  2     2    46987   30    48     4         1  198    395     7       0
  3     3    28994   63    81     4         1  310    503     7       0
  4     4    16665   18    36     4         1  117    267     7       0
  5     5     8011  114   114     4         1  524    524     6       0
  6     6    15276   81    99     4         1  275    377     6       0
  7     7    23223  141   159     4         1  353    504     6       0
  7     7    ERRORS DECODED IN POSITIONS 328, 329, 330, 332, 333, 336, 339, 340
  8     1     1301  147   150     4        85  524    524     5       0
  9     2    13710   33    51     4       189  348    465     5       0
 10     3     9262    3    21     4       301  311    423     5       1
 11     4     2023  174   192     4       108  295    351     5       0
 12     6     4717   21    39     4       266  292    387     5       0
 13     7    11577   -3    15     4       344  354    467     5       1
 14     2    13562   -6    12     4       339  348    465     5       2
 15     3     7828    3    21     4       302  311    423     5       3
 16     4     2199    3    21     4       286  295    351     5       4
 17     6     7220    3    21     4       283  291    405     5       5
 18     7   213150   81   105     5         1  214    502     5       0
 19     2   235821   36    60     5         1  195    403     5       0
 20     3    15744  120   123     5         1  524    524     4       0
 21     4    10008  159   159     4         1  524    524     3       0
 22     6     4868  198   219     4         1  222    303     3       0
 23     7     4777   78    78     4       205  524    524     2       0
 24     2     1104  144   144     4       186  524    524     1       0
 25     6      312  312   312     4       213  524    524     0       0

     ERROR RATE BY TRACKS 0.0914, 0.1105, 0.0924, 0.1010, 0.0857, 0.0924, 0.0819
     TOTAL COMPUTATIONS THIS BLOCK  747655
```

Fig. 8 - Sample program output.

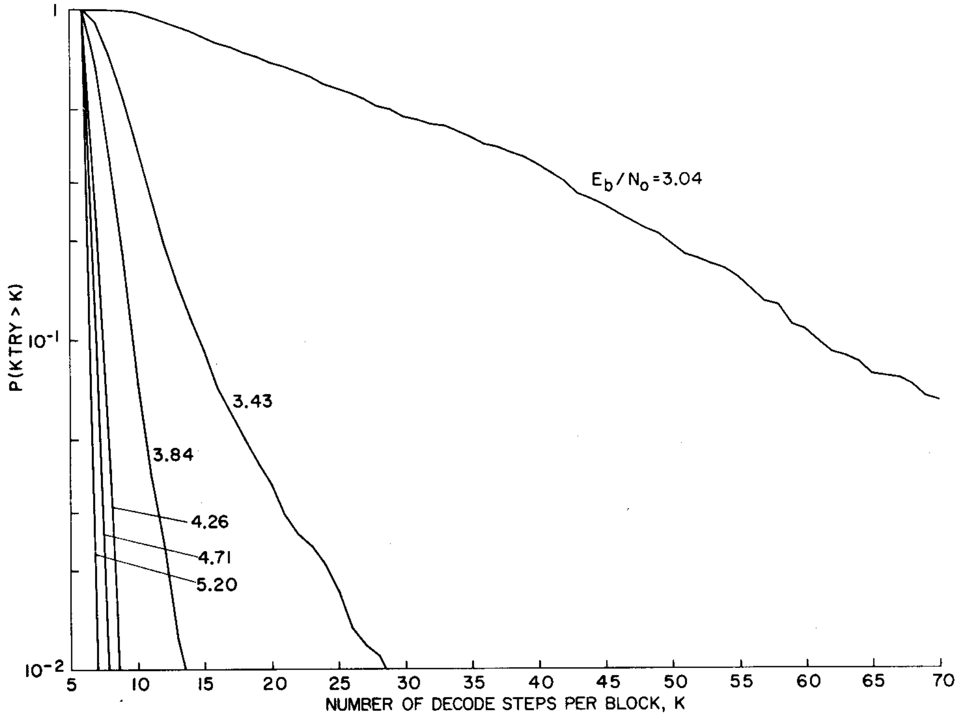
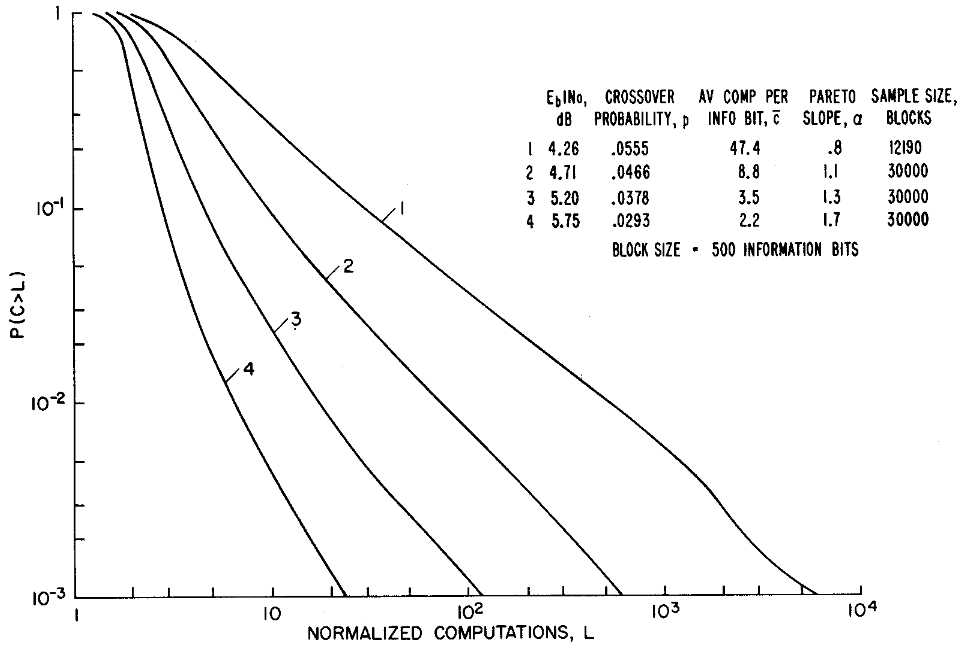


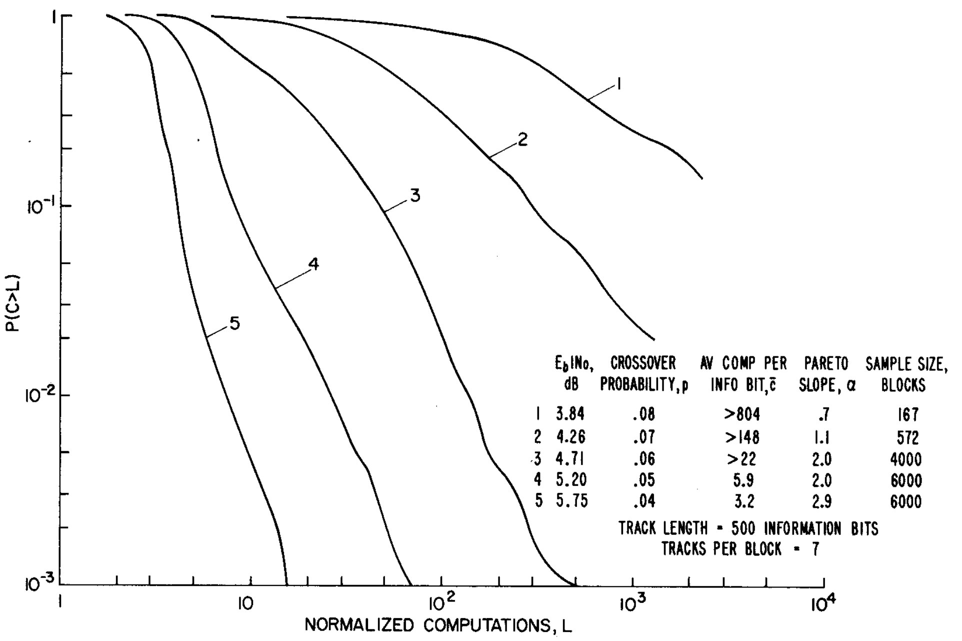
**Fig. 9 - Probability that the number of decode steps will exceed K for the optimized pull-up decoder as a function of $E_b/N_o$.**



|  | EbINo, dB | CROSSOVER PROBABILITY, p | AV COMP PER INFO BIT, $\bar{c}$ | PARETO SLOPE, $\alpha$ | SAMPLE SIZE, BLOCKS |
|---|---|---|---|---|---|
| 1 | 4.26 | .0555 | 47.4 | .8 | 12190 |
| 2 | 4.71 | .0466 | 8.8 | 1.1 | 30000 |
| 3 | 5.20 | .0378 | 3.5 | 1.3 | 30000 |
| 4 | 5.75 | .0293 | 2.2 | 1.7 | 30000 |

BLOCK SIZE = 500 INFORMATION BITS

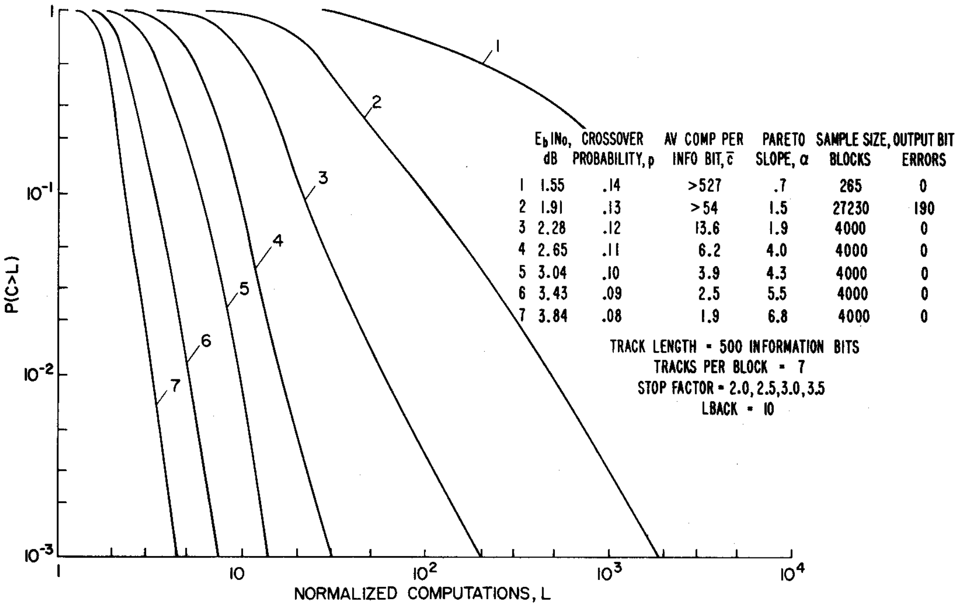
**Fig. 10 -Simple sequential decoder computations performance for the BSC as a function of $E_b/N_o$.**



| | $E_b/N_o$, dB | CROSSOVER PROBABILITY, p | AV COMP PER INFO BIT, $\bar{c}$ | PARETO SLOPE, $\alpha$ | SAMPLE SIZE, BLOCKS |
|---|---|---|---|---|---|
| 1 | 3.84 | .08 | >804 | .7 | 167 |
| 2 | 4.26 | .07 | >148 | 1.1 | 572 |
| 3 | 4.71 | .06 | >22 | 2.0 | 4000 |
| 4 | 5.20 | .05 | 5.9 | 2.0 | 6000 |
| 5 | 5.75 | .04 | 3.2 | 2.9 | 6000 |

TRACK LENGTH = 500 INFORMATION BITS
TRACKS PER BLOCK = 7

**Fig. 11 - Pseudo Falconer decoder computations performance for the BSC as a function of $E_b/N_o$.**



| | $E_b/N_o$, dB | CROSSOVER PROBABILITY, p | AV COMP PER INFO BIT, $\bar{c}$ | PARETO SLOPE, $\alpha$ | SAMPLE SIZE, BLOCKS | OUTPUT BIT ERRORS |
|---|---|---|---|---|---|---|
| 1 | 1.55 | .14 | >527 | .7 | 265 | 0 |
| 2 | 1.91 | .13 | >54 | 1.5 | 27230 | 190 |
| 3 | 2.28 | .12 | 13.6 | 1.9 | 4000 | 0 |
| 4 | 2.65 | .11 | 6.2 | 4.0 | 4000 | 0 |
| 5 | 3.04 | .10 | 3.9 | 4.3 | 4000 | 0 |
| 6 | 3.43 | .09 | 2.5 | 5.5 | 4000 | 0 |
| 7 | 3.84 | .08 | 1.9 | 6.8 | 4000 | 0 |

TRACK LENGTH = 500 INFORMATION BITS
TRACKS PER BLOCK = 7
STOP FACTOR = 2.0, 2.5, 3.0, 3.5
LBACK = 10

**Fig. 12 - Pull-up decoder computations performance for the octal channel as a function of $E_b/N_o$.**