Department of

ELECTRICAL ENGINEERING

UNIVERSITY OF NOTRE DAME, NOTRE DAME, INDIANA

ANNUAL STATUS REPORT

Error Control Coding Techniques for Space and Satellite Communications

NASA Grant Number NAG5-557
Report Number 97-001

Principal Investigator: Daniel J. Costello, Jr.
Graduate Assistants: Hermano A. Cabral
Jiali He

Department of Electrical Engineering
University of Notre Dame

March 19, 1997

Part I

Iterative Bootstrap Hybrid Decoding Using the Multiple Stack Algorithm

Introduction

Bootstrap Hybrid Decoding (BHD) (Jelinek and Cocke, 1971) is a coding/decoding scheme that adds extra redundancy to a set of convolutionally encoded codewords and uses this redundancy to provide reliability information to a sequential decoder.

Theoretical results indicate that bit error probability performance (BER) of BHD is close to that of Turbo-codes, without some of their drawbacks.

In this report we study the use of the *Multiple Stack Algorithm* (MSA) (Chevillat and Costello, Jr., 1977) as the underlying sequential decoding algorithm in BHD, which makes possible an iterative version of BHD.

Sequential Decoding

Sequential Decoding is a decoding method for convolutional codes. It searches the most likely path based on some metric, typically the *Fano metric*, which takes into account the difference in path lengths when comparing two paths.

Its main advantage is the relative independence of the computational complexity with respect to the constraint length of the code, which allows for very large constraint lengths and very low BER's.

It has, however, a variable number of computations per decoded bit, which can cause *erasures*, i.e., a decoding time long enough to cause information loss, particularly at rates close to the channel *cut-off rate*, R_0.

The Stack Algorithm

The two best known sequential decoding algorithms are the *Fano algorithm* and the *Stack algorithm*.

The Stack algorithm is based on an ordered list of paths, the stack, with the top path in the stack being extended during each decoding cycle, and its successors being inserted (in order) in the stack. The algorithm ends when the top path reaches the end of the tree.

The only other way to end the decoding process is when the decoding time becomes too large, causing the input buffer to overflow.

The Multiple Stack Algorithm

The *Multiple Stack Algorithm* (MSA) is a sequential decoding algorithm based on the Stack algorithm. The two algorithms are identical until the stack fills up. While the Stack algorithm just discards the path in the stack with the smallest metric, the MSA creates an additional stack, transfers the T (a user-specified number) topmost paths in the preceding stack, and continues decoding in the new stack.

This pattern is repeated until the top path in the current stack reaches the end of the tree. If the current stack is not the first stack, the top path's information bits are taken as a tentative decision and are stored together with the corresponding metric. The decoder then returns to work in the immediately preceding stack.

The algorithm ends if either the end of tree is reached in the first stack, or if some maximum number of computations (a user-specified number) is performed, in which case the best tentative decision is taken as the decoded path.

The MSA has the advantages of eliminating erasures (at the expense of a higher BER) and having a lower computational variability (at the expense of a higher average number of computations).

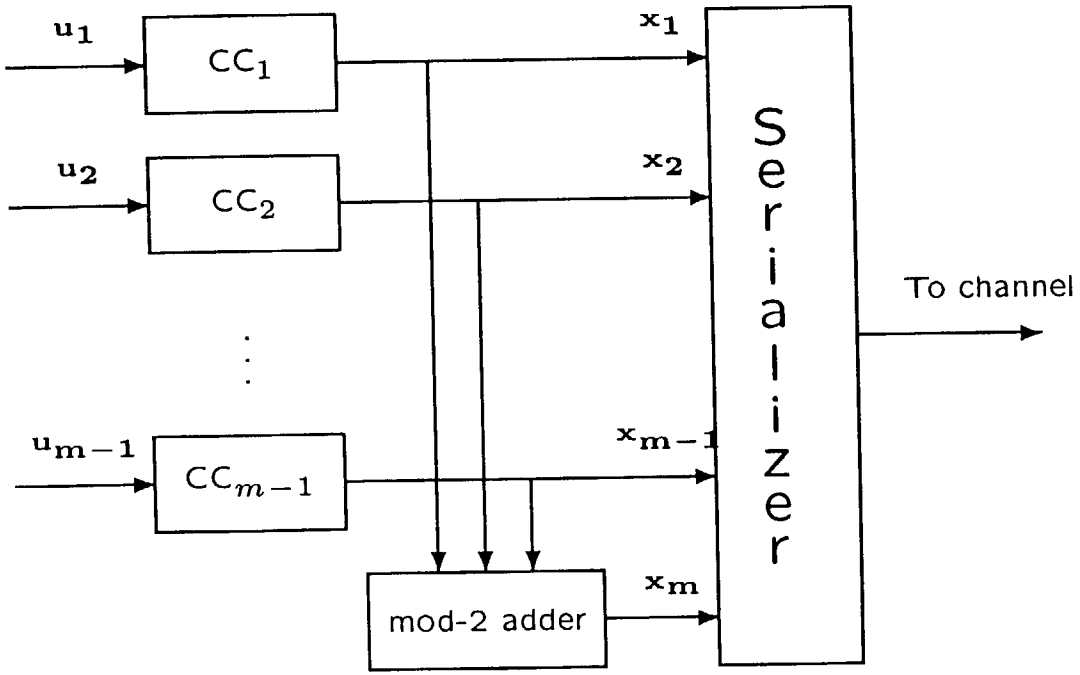
Bootstrap Hybrid Decoding (BHD)

An interesting variation of sequential decoding was presented in 1971 by Jelinek and Cocke, called *Bootstrap Hybrid Decoding* (BHD). The idea was to provide some additional information to the sequential decoder to help it proceed through the tree and reduce the computational variability.

This additional information was provided by a parity check sequence taken across a set of convolutional codewords, which is also transmitted over the channel.

Upon receiving the codewords and the parity sequence, the decoder generates a *channel state stream* containing all the information provided by the extra redundancy and uses it to adjust the metric used by the sequential decoder.

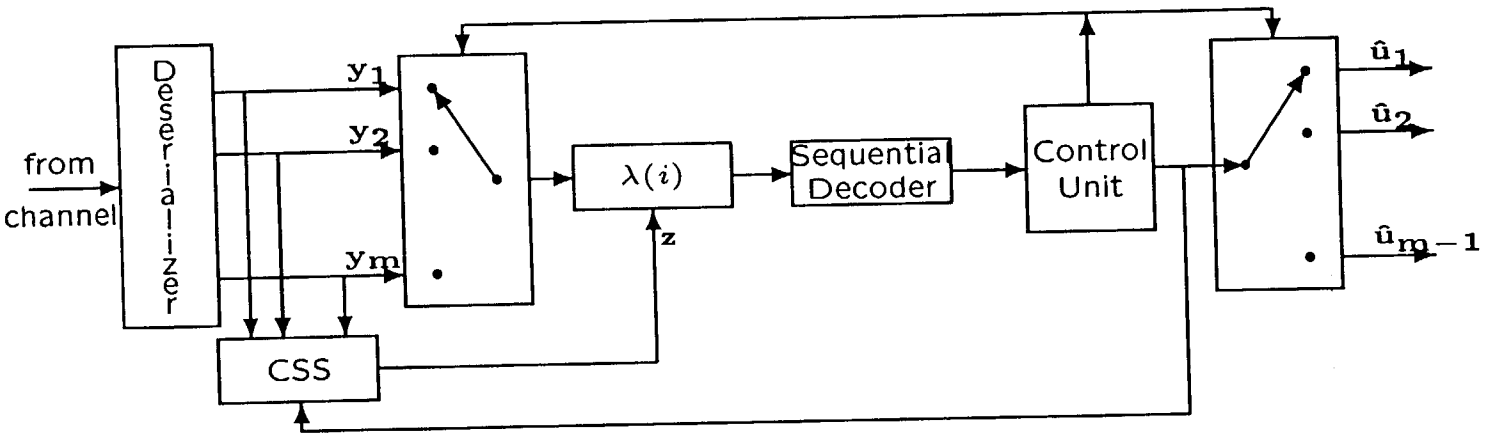
BHD Encoder


u_1
CC_1
x_1
u_2
CC_2
x_2
Serializer
To channel
u_{m-1}
CC_{m-1}
x_{m-1}
mod-2 adder
x_m

BHD (cont'd)

The decoder tries to decode one stream at a time, using the adjusted metric. Upon successful decoding of one stream, the estimated information sequence is used to update the channel state stream, and the pattern repeats until only one stream remains to be decoded, which can be found from the other streams and the parity constraint.

BHD Decoder


from
channel
Deserializer
y_1
y_2
y_m
$\lambda(i)$
z
Sequential
Decoder
Control
Unit
CSS
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_{m-1}$

Using the MSA

Frequent channel state stream updates are a desirable feature in the BHD scheme, for the smaller the number of streams checked by the channel state stream, the stronger is its help in decoding those streams.

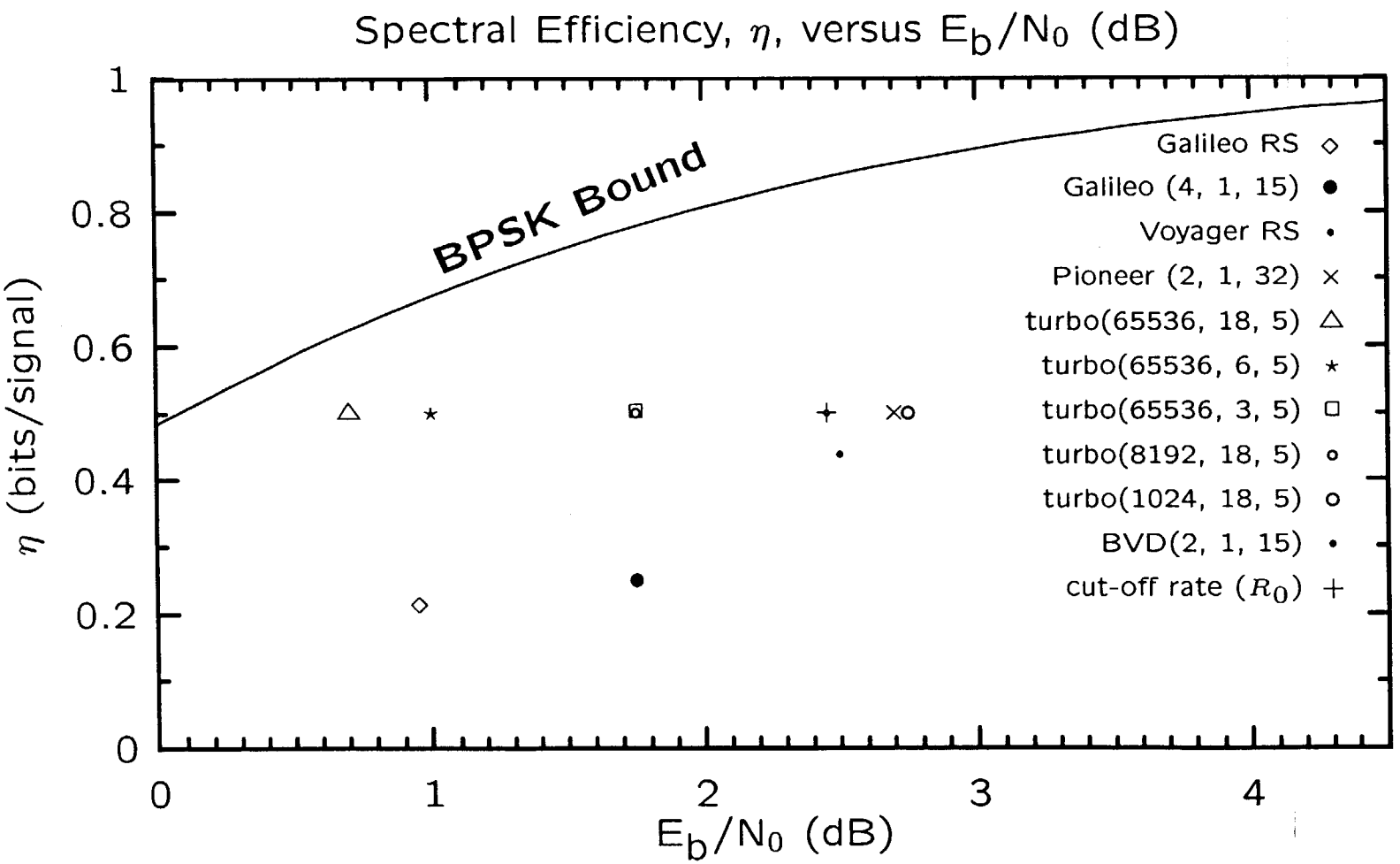
However, one can only update the channel state stream after successfully decoding a stream, or at least the initial part of a stream in the case of incomplete decoding (when an improved version, called the pullup algorithm, of the BHD is being used).

Faster updates can be achieved by using the MSA. The idea is to use the tentative decisions provided by the MSA in the case of incomplete decoding to update the channel state stream for those positions considered to be 'reliable'. If a stream is successfully decoded, the update is done as before.

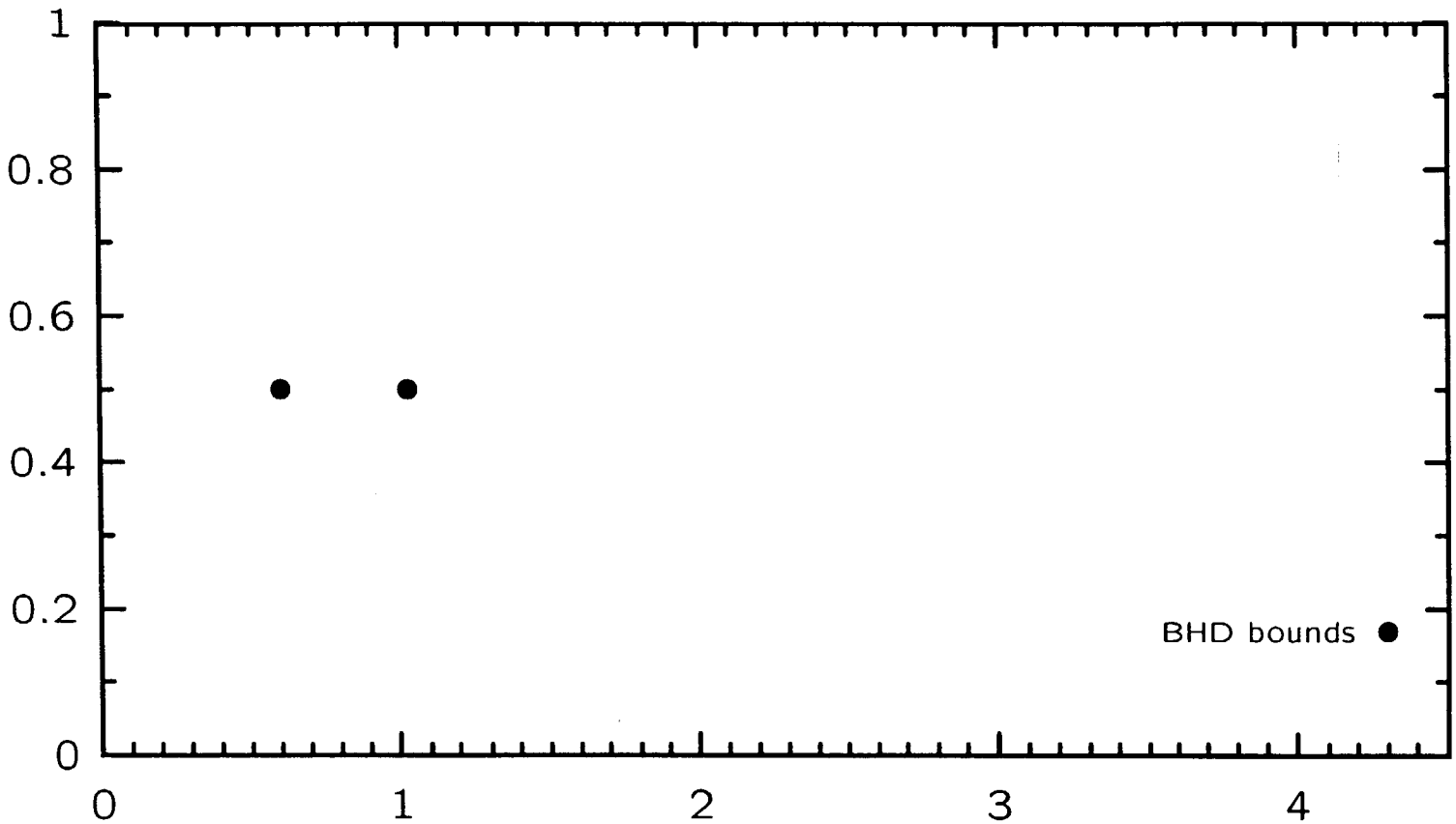
BHD Bounds

Upper and lower bounds to the set of rates for which the average number of computations is bounded for the BHD scheme can be derived. These bounds bracket the performance of Turbo-codes, thus suggesting that performance similar to Turbo-codes can be obtained with the BHD scheme.

BHD Bounds


Spectral Efficiency, η, versus E_b/N_0 (dB)
η (bits/signal)
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
E_b/N_0 (dB)
BPSK Bound
Galileo RS
Galileo (4, 1, 15)
Voyager RS
Pioneer (2, 1, 32)
turbo(65536, 18, 5)
turbo(65536, 6, 5)
turbo(65536, 3, 5)
turbo(8192, 18, 5)
turbo(1024, 18, 5)
BVD(2, 1, 15)
cut-off rate (R_0)

BHD Bounds



Part II

Soft-Output Sequential Decoder

Introduction

Most decoding algorithms deliver an estimate of the transmitted codeword after processing the received vector, with no further information on the reliability of the estimate. These are known as "hard-output" decoding algorithms.

It has recently become clear that "soft-output" decoding algorithms can give significant gains in performance, especially when used in concatenated or iterative decoding schemes, such as Turbo-codes. Examples of such algorithms are the MAP algorithm and the Soft-Output Viterbi Algorithm (SOVA).

MAP Algorithm

The MAP (Maximum A Posteriori) algorithm (Bahl, et al, 1974) gives the probabilities of the states and transitions of a Markov chain at each time unit, conditioned on the received symbol vector. When applied to the decoding of convolutional codes, the algorithm outputs two vectors of probabilities, corresponding to the probabilities of the input and output symbols.

The MAP algorithm is a forward-backward algorithm. It starts through the code trellis in the forward direction, computing some of the parameters needed, and after reaching the end of the trellis, goes backwards computing the remaining parameters.

One disadvantage of MAP decoding is that, at every time unit, the decoder has to store not only the metrics of the survivors at every state but also the parameters needed to compute the state probabilities at the end. This involves significant increases in storage compared to the standard Viterbi algorithm.

The Soft-Output Viterbi Algorithm (SOVA)

The SOVA also outputs reliability information on the decoder estimate, but it does so with less computational effort (and accuracy) than the MAP algorithm.

At every time unit, the algorithm computes the survivor at each state, just like the ordinary Viterbi algorithm, and based on the reliability of this survivor decision (which is basically the difference between the metrics of the competing paths), updates a vector of probabilities (one for each state). This vector gives the probabilities that the survivor bits are incorrect.

At the end of the trellis search, the reliability information consists of the vector of probabilities corresponding to the decoded path.

The advantage of the SOVA over the MAP algorithm is less computational effort and storage.

A Soft-Output Sequential Decoder

A similar idea to the MAP algorithm can be used to come up with a Soft-Output sequential decoder. The usefulness of such a decoder resides in applications to iterative schemes like *Bootstrap Hybrid Decoding* (BHD) or a Turbo-code-like scheme using sequential decoding and large constraint length codes.

Since sequential decoders work based on a tree code, instead of a trellis, the algorithm compute the probabilities for each node and each branch in the tree, and from those derives the probabilities for the input and output symbols.

However, unlike the MAP algorithm, a sequential decoder doesn't traverse the whole tree, and therefore some of the node and branch probabilities must be computed without knowing the actual output branch symbols. This is done assuming a random tree code where the symbols are equally likely. These values, however, are not stored, but only used to compute the parameters for the visited nodes.

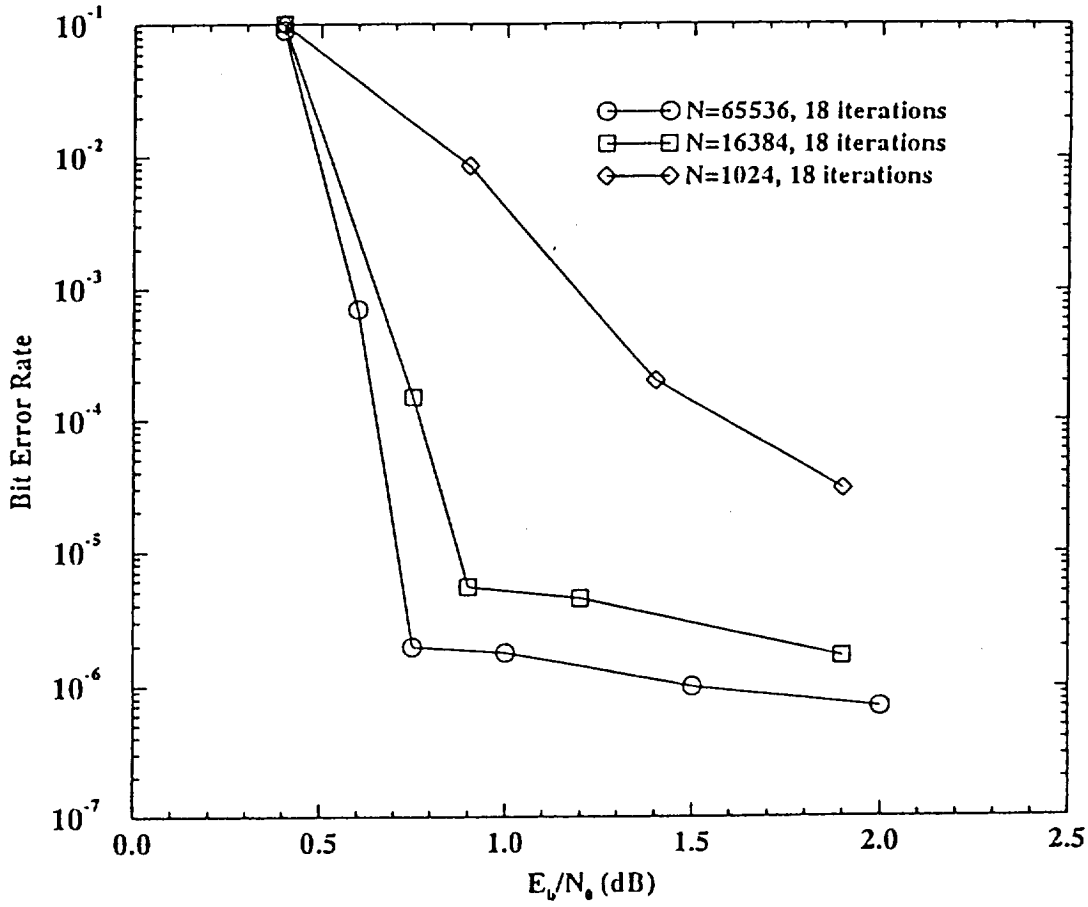
In the same way as the MAP algorithm, it's necessary to traverse the tree in both directions, forward and backward, to be able to compute all necessary parameters. The backward search, however, only travels through the already visited nodes.

After the forward and backward searches, the node and branch probabilities can be computed, and from them the input and output symbol probabilities. Note, however, that since the whole tree hasn't been examined, these probabilities are only approximations to the true probabilities, just like the estimate of a standard sequential decoder is only an approximation (albeit a good one) to the true Maximum Likelihood solution.

Part III

Concatenated Turbo Codes

Performance of Turbo Codes



- **Turbo codes have an overall distance spectrum similar to a random code, and thus they are capable of superior performance on noisy channels. With a block length of 2^{16}, they can achieve a bit error rate (BER) of 10^{-5} at a channel signal-to-noise ratio (SNR) only 0.7 dB away from capacity, an improvement of about 2 dB compared to the best previously known codes. Furthermore, an iterative MAP decoding algorithm allows this remarkable performance to be achieved with moderate decoding complexity.**

- **Problems with Turbo codes:**

--Due to their small free distance, there seems to be no way to eliminate the error floor by increasing the block size or improving the interleaver.

--Near capacity performance requires considerable decoding effort, e.g., to achieve a BER of 10^{-5} only 0.7 dB away from capacity, 18 iterations of MAP decoding are required.

--Performance depends on the interleaver size, thus resulting in a long delay.

- **We have investigated various concatenation schemes with Turbo codes in an attempt to achieve some or all of the following:**

--Eliminate the error floor.

--Achieve similar performance with less decoding effort (fewer iterations of decoding).

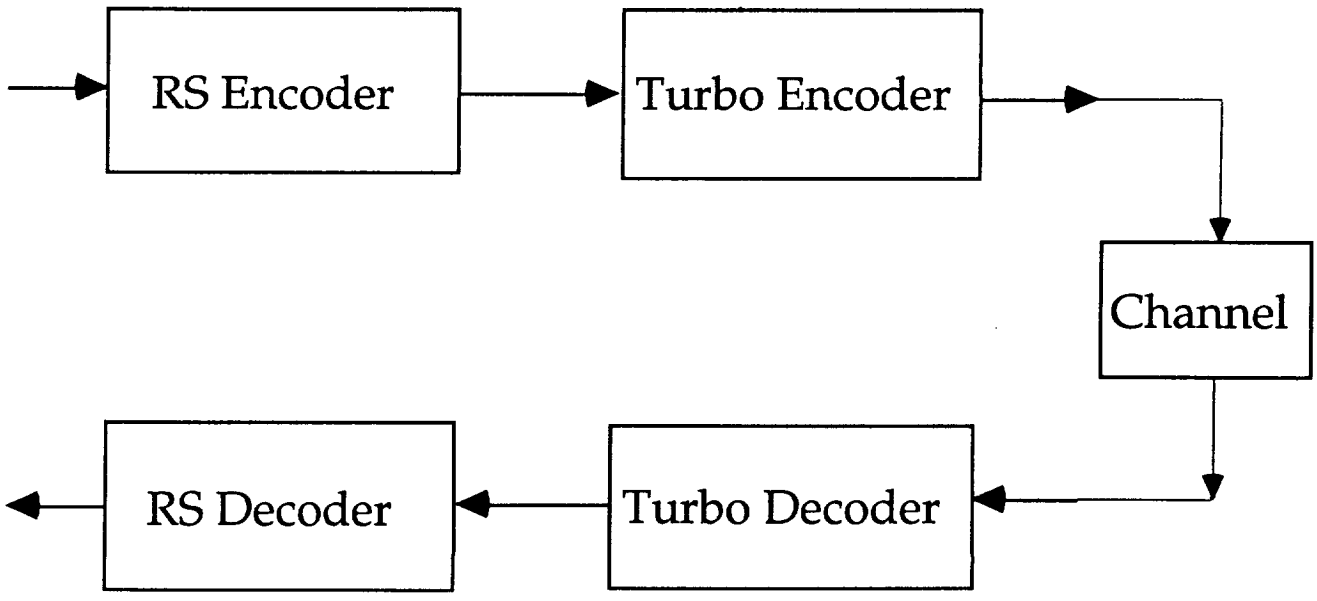
--Achieve similar performance with smaller interleaver size (reduced delay).

Lowering the Error Floor

- **The presence of an error floor is due to the Turbo codes' small free distance. A small number of errors, usually a small even number of errors in a block, can often 'fall through' the iterative decoding process and not be corrected.**

- **Concatenating a Turbo code with an outer code, which is optimized for free distance, can lower the error floor. The outer code will pick up the residual small number of errors and correct them.**

- **A concatenation scheme with an inner Turbo code and an outer Reed-Solomon code:**



- **Estimated BER of this scheme:**

$$p_b < \frac{2^{\tilde{K}-1}}{2^{\tilde{K}}-1} \sum_{j=t+1}^{n} \frac{j+t}{n} \binom{n}{j} p_s^{\,j} (1-p_s)^{n-j} \qquad (1)$$

p_b --bit error rate at the RS decoder output

p_s --symbol error rate at the RS decoder input

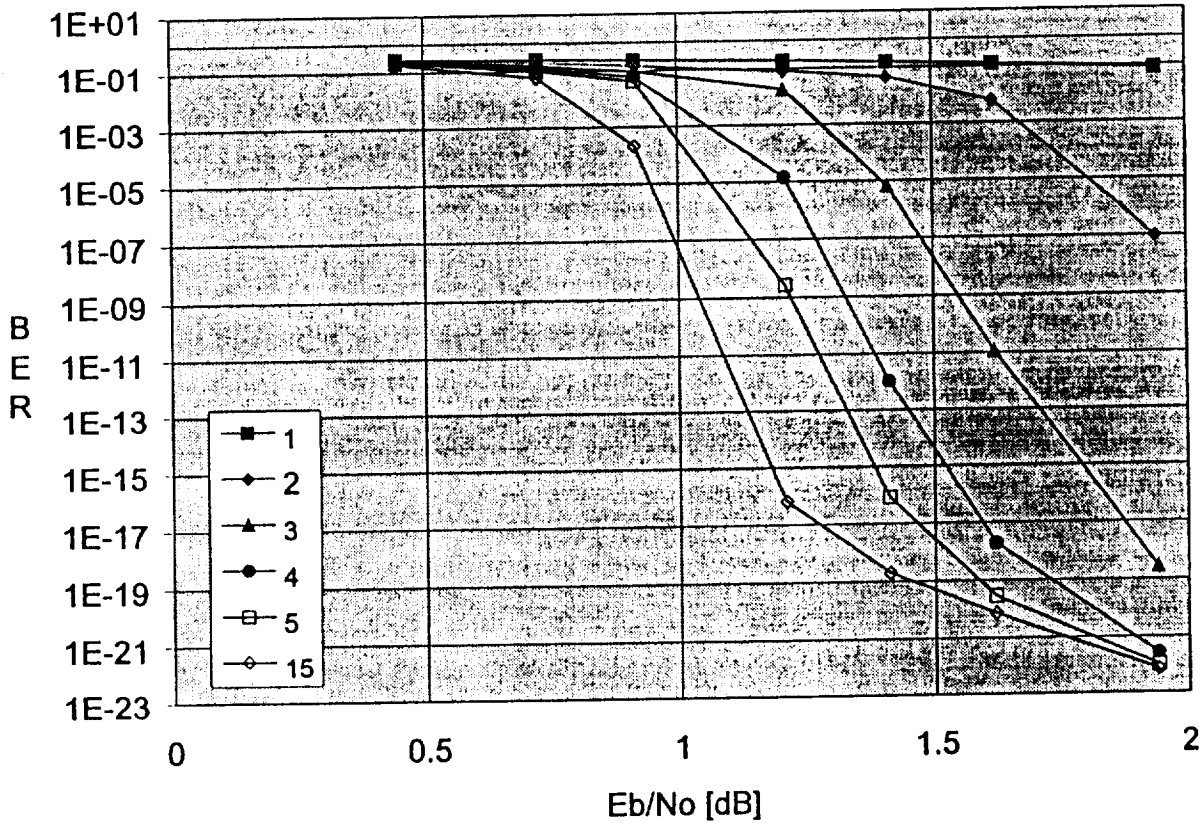
n --block length of RS code in symbols

$\tilde{K}$ --number of bits per symbol

t --error correcting capability of RS code

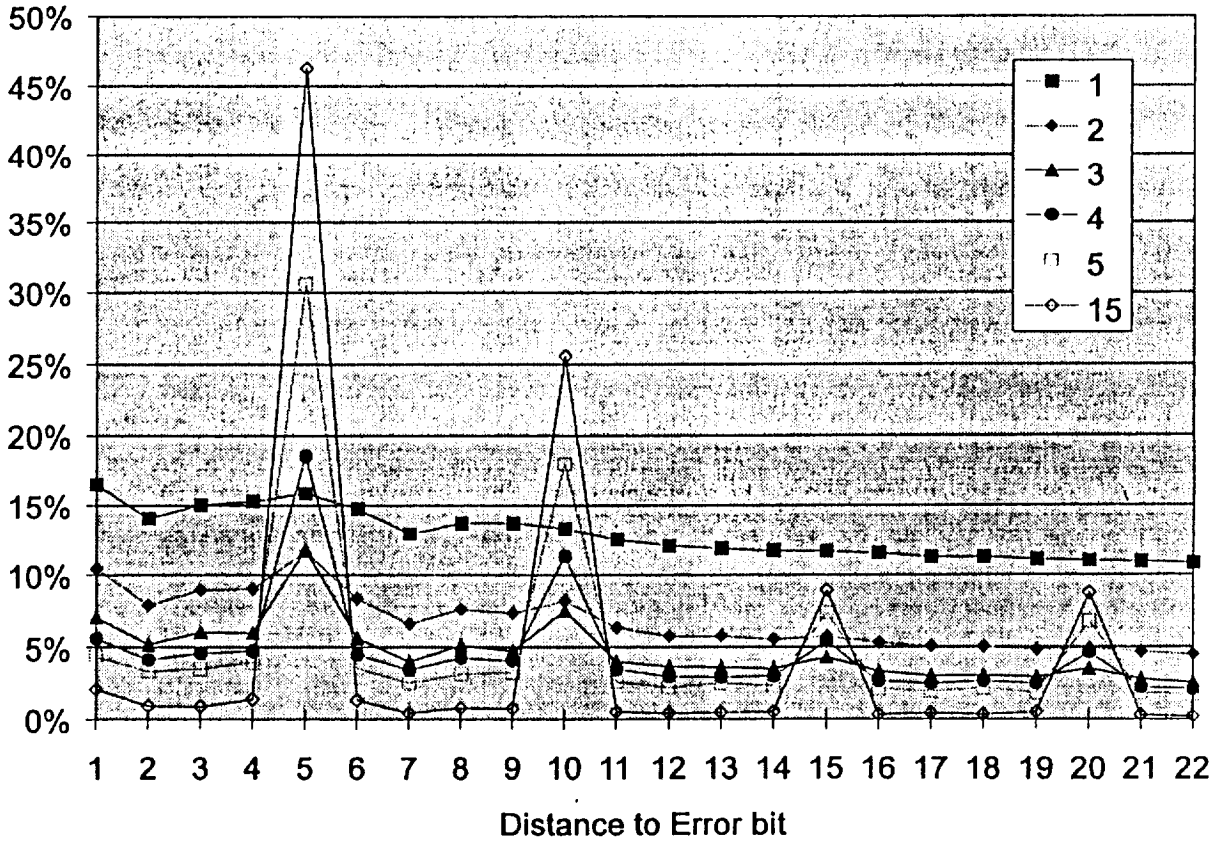
We assume a $t = 8$, $\tilde{K} = 8$, $n = 255$, i.e., a (255, 239) RS code, in the following.

- **Based on a simulation of Turbo codes** (block size=255x8=2040), **the bounds were calculated using the above formula and showed a large decrease in the error floor.**

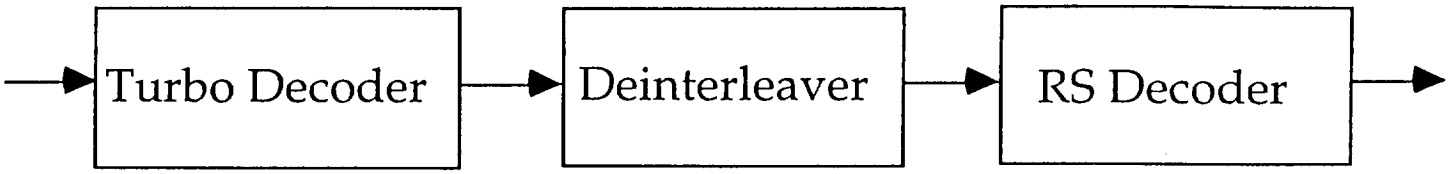

1E+01
1E-01
1E-03
1E-05
1E-07
1E-09
1E-11
1E-13
1E-15
1E-17
1E-19
1E-21
1E-23
BER
0
0.5
1
1.5
2
Eb/No [dB]
1
2
3
4
5
15

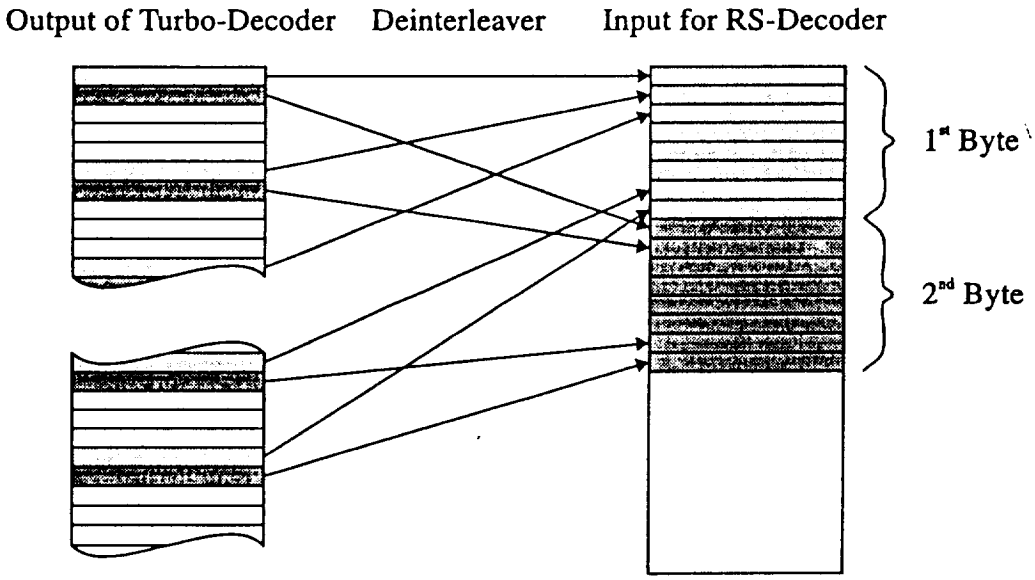
Improvement with an Intra-code Bit Interleaver

- **The error distribution of the decoded bits within a Turbo code block has some structure. The distance between two or more errors is not uniformly distributed. If there is an error at position k, then the probability of another error at position $k + j \cdot n_{cycle}$, where n_{cycle} is the cycle length of the recursive convolutional code, is high.**

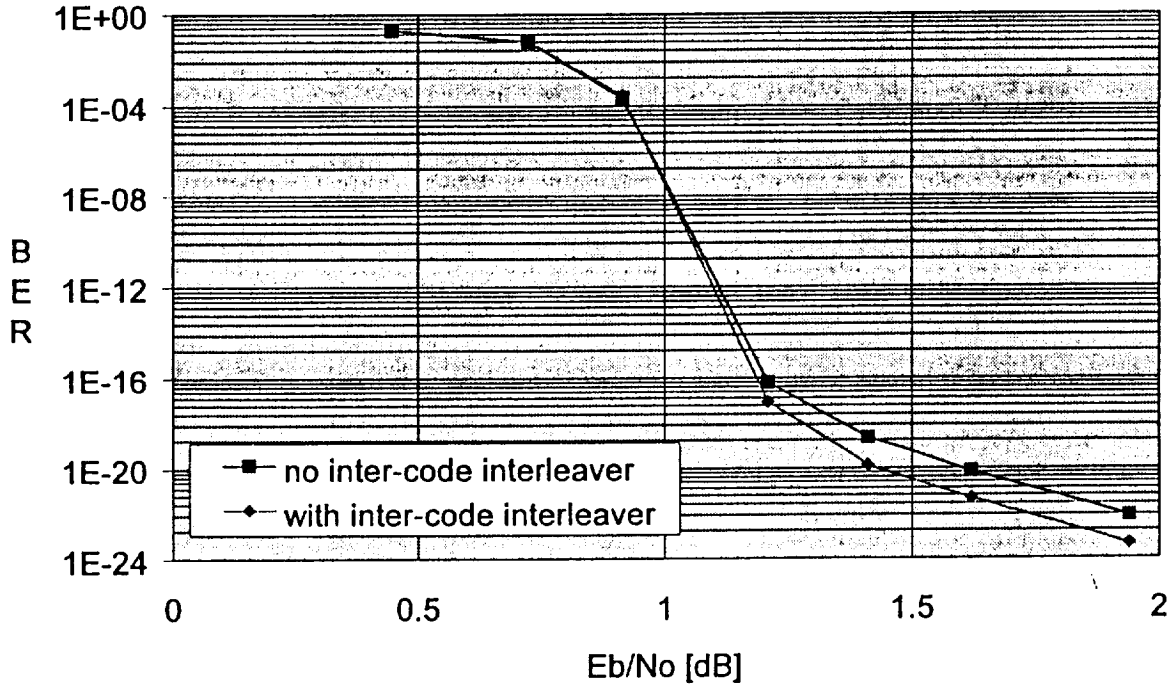


- **In the previous concatenation scheme, we can make use of this fact to decrease the symbol error rate at the input to the RS decoder:**

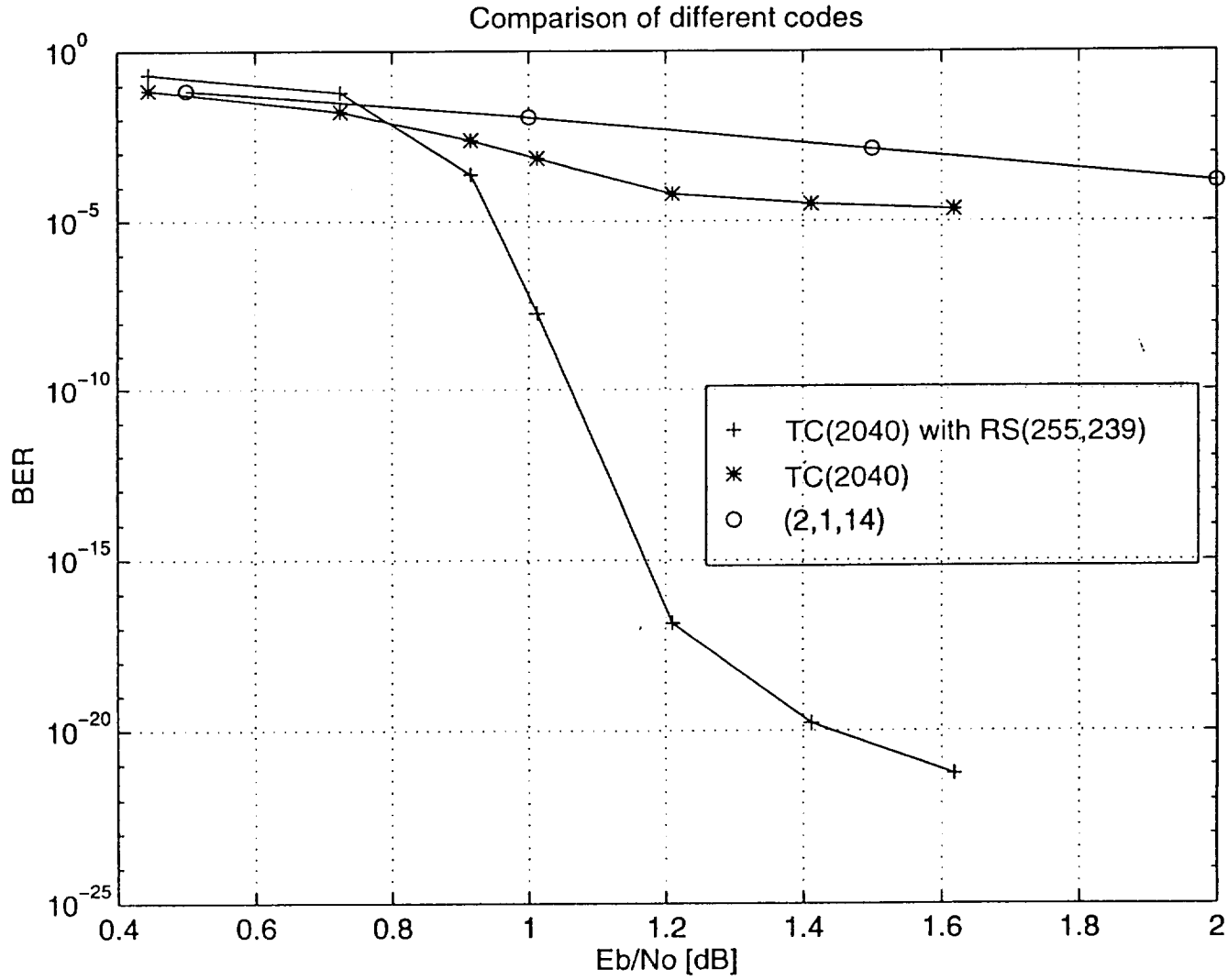




- **Bounds calculated with the intra-code bit interleaver show improvement(not much, though):**



- **Thus with an outer (255,239) Reed-Solomon code, the error floor can be lowered significantly while the overall code rate only decreases from 0.50 to 0.47.**



Further Discussion of These Results

- **Formula (1), which was used to estimate the BER of the concatenation scheme, assumes that the symbol errors are random, i.e., ideal symbol interleaving between the inner and outer codes. However, extensive simulations show that this is not the case with Turbo codes.**

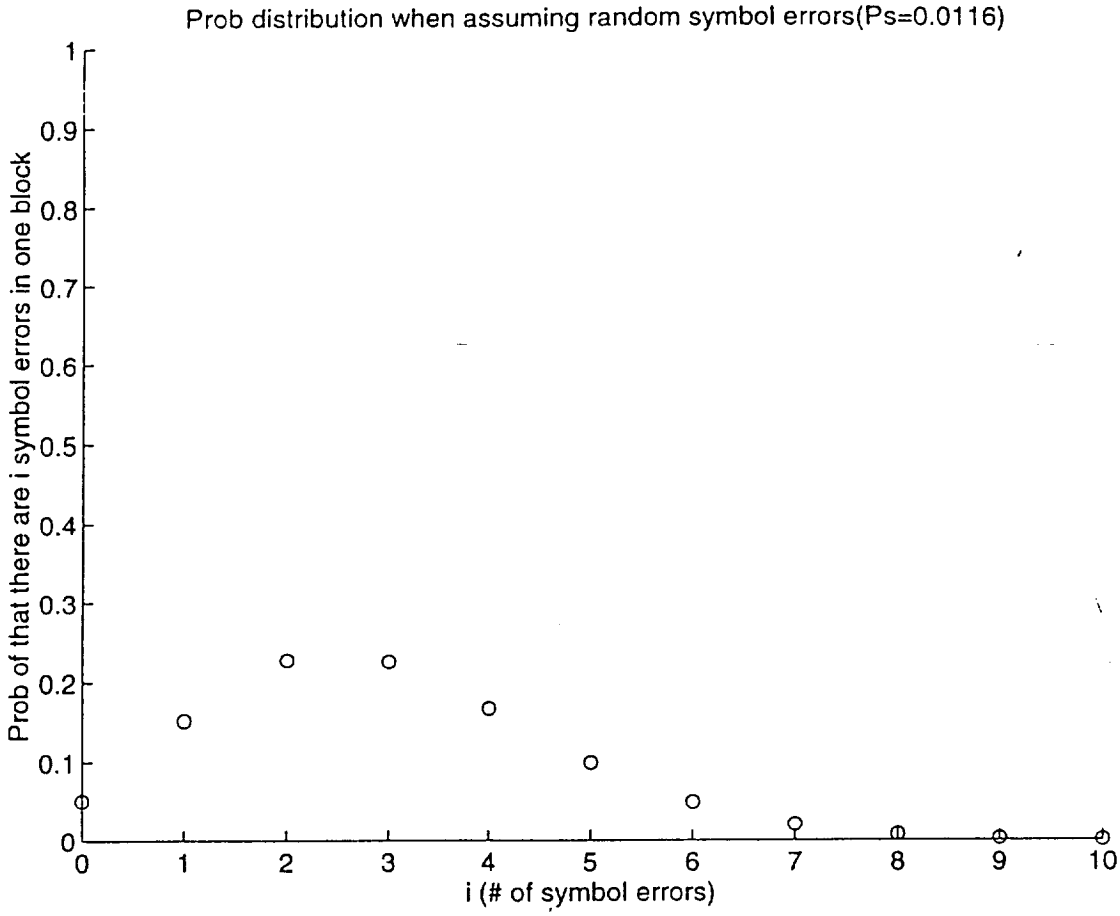
- **Assuming random symbol errors, and a symbol error rate = p_s, then the probability that there are i symbol errors in a block is:**

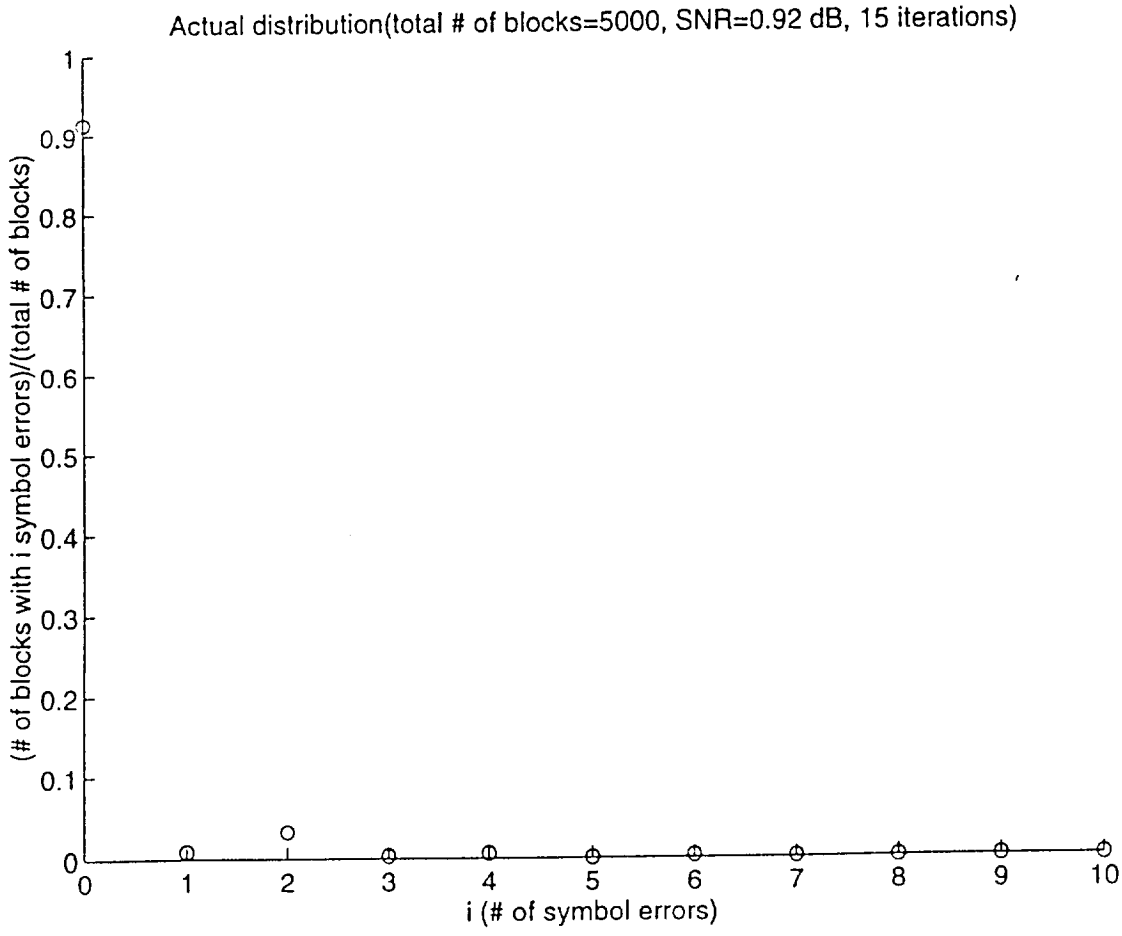
$$\binom{n}{i} p_s^{\,i} (1 - p_s)^{255-i} \qquad (2)$$

and when p_s is small, the probability that there are many symbol errors in a block is very small.

- **On the other hand, we can calculate the relative frequency of i symbol errors in a block, $R.F.(i)$, based on simulation results:**

$$R.F.(i) = \frac{\#\ of\ blocks\ with\ i\ symbol\ errors}{total\ \#\ of\ blocks} \qquad (3)$$


Prob distribution when assuming random symbol errors(Ps=0.0116)
Prob of that there are i symbol errors in one block
i (# of symbol errors)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
1
2
3
4
5
6
7
8
9
10


Actual distribution(total # of blocks=5000, SNR=0.92 dB, 15 iterations)
(# of blocks with i symbol errors)/(total # of blocks)
i (# of symbol errors)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
1
2
3
4
5
6
7
8
9
10

- **Clearly the symbol errors at the input of the RS decoder are not randomly distributed. Take a close look at the simulation results:**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4570	42	156	14	28	4	13	6	3	3	2	2	1	1	5
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	0	0	0	1	1	1	0	0	1	0	0	0	2	2
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
0	0	1	0	0	0	2	0	3	0	0	0	1	1	0
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
0	1	1	2	0	0	1	0	1	0	0	0	0	0	2
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
0	0	1	0	0	2	0	0	0	0	1	2	0	1	1
75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
0	1	3	1	1	3	2	0	0	5	0	4	1	1	1
90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
2	1	3	1	2	2	3	1	1	3	2	4	3	2	1
105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
2	3	1	3	0	3	1	3	4	2	4	1	5	1	3
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134
1	2	0	2	1	1	0	4	3	2	3	0	0	1	0
135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
0	1	0	0	0	0	1	2	0	0	0	0	0	0	1

.....

- **For Turbo codes, most blocks are error free. There are a lot of blocks with 2 symbol errors(caused by 2 bit errors). There are a few blocks with many symbol errors, which is almost impossible when assuming random symbol errors.**

- **In the above example, the blocks which have more than 8 symbol errors cannot be corrected by the outer RS code, which means the BER bounds obtained using formula (1) are better than the actual performance.**

- **Bit interleaving can decrease the symbol error rate at the input to the RS decoder, but it doesn't help much for the blocks with many symbol errors.**

# of blocks = 3000, SNR=1.21 dB, 5 iterations

Without bit interleaving:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2703	48	104	42	30	16	8	2	4	3	2	0	1	4	3
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
2	1	1	2	2	0	0	2	1	1	0	0	3	2	0
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
0	0	1	0	0	1	0	0	1	1	0	0	1	0	0
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
0	0	1	0	0	0	0	0	0	1	0	0	0	1	0
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
0	0	0	1	0	2	0	0	0	0	0	0	0	1	0
75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

total # of symbol errors = 1889

# of blocks with more than 8 symbol errors = 43

With bit interleaving:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2703	118	64	33	21	11	5	2	3	1	4	2	2	2	2
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	0	0	3	0	3	1	0	1	0	1	2	0	1	1
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
1	0	0	2	0	0	0	0	0	0	0	0	0	0	0
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
1	0	2	0	0	0	0	0	0	2	0	1	0	0	0
75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

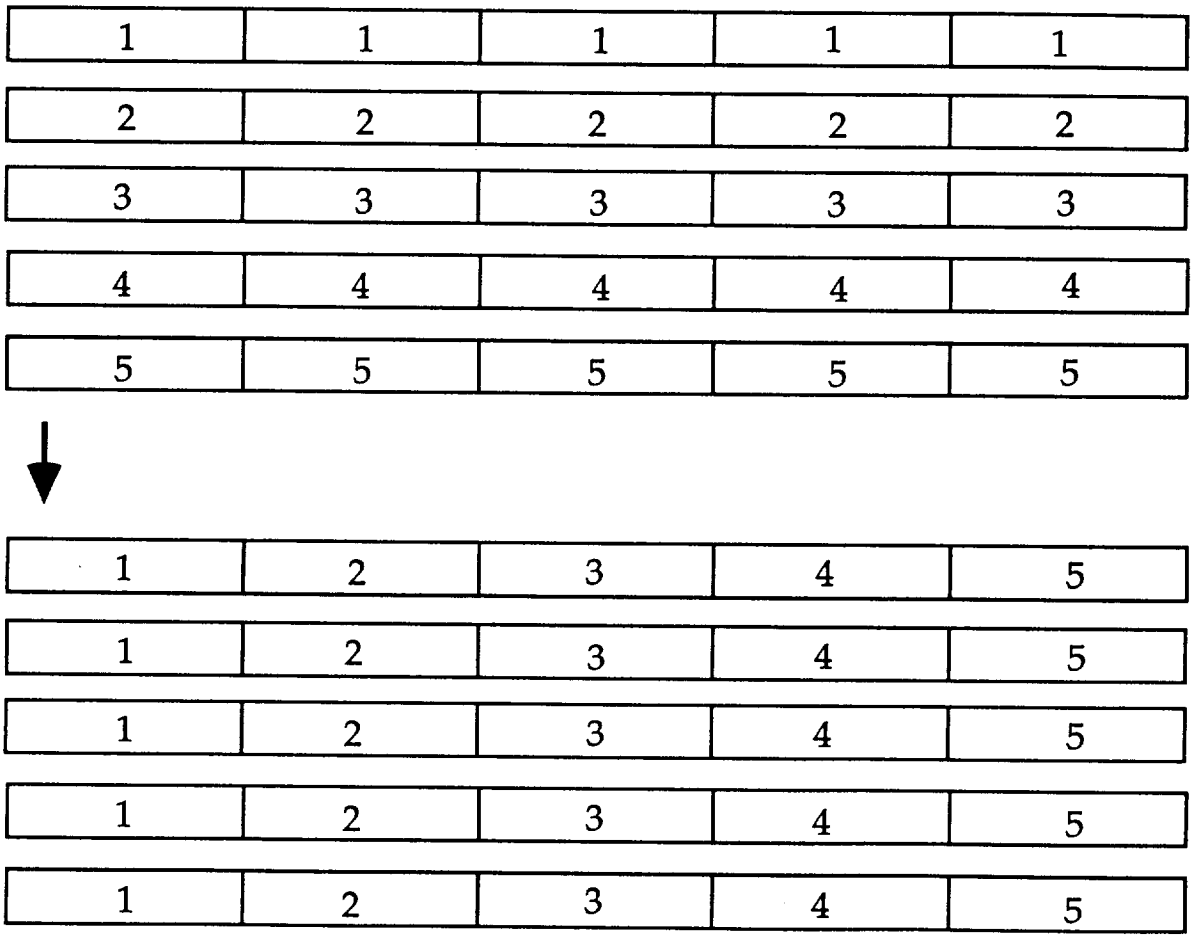
total # of symbol errors = 1737

# of blocks with more than 8 symbol errors = 40

Using Symbol Interleaving

- **Symbol interleaving can be used to spread the many symbol errors in one block to several blocks, thus making the symbol errors more random and generating more correctable blocks.**

Interleaving depth = 5



# of blocks=21000, SNR=1.21 dB, 5 iterations

Interleaving depth = 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
19041	401	738	202	168	81	45	42	26	18	17	16	13	15	12

15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
12	6	6	9	8	7	6	3	5	7	5	3	6	3	3

30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
1	1	4	1	0	4	2	2	4	3	3	2	2	0	4

45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
0	3	1	0	0	2	2	0	0	4	2	1	1	1	0

.....

# of blocks with more than 8 symbol errors = 256

Interleaving depth = 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
16541	2617	773	309	173	110	79	71	71	42	34	33	24	23	16

15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
12	10	16	11	6	5	10	3	0	1	2	2	2	0	1

30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

# of blocks with more than 8 symbol errors = 256

Interleaving depth = 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
15042	3596	1146	484	273	169	123	82	36	23	15	6	1	1	1

15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0

# of blocks with more than 8 symbol errors = 49

# of blocks = 21000, SNR =1.21 dB, 15 iterations

Interleaving depth = 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
20257	105	522	27	59	8	13	2	1	0	1	1	0	1	0

15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

.....		133		.....
0		1		0

<u># of blocks with more than 8 symbol errors = 6</u>

Interleaving depth = 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
19515	1319	138	13	1	1	1	1	1	0	0	2	1	2	0

15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	0	0	0	0	0	0	1	1	0	0	2	0	0

30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

<u># of blocks with more than 8 symbol errors = 10</u>

Interleaving depth = 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
19422	1442	9	11	7	3	1	4	2	2	4	2	0	1	0

<u># of blocks with more than 8 symbol errors = 9</u>

- For 5 iterations of decoding, symbol interleaving gives significant improvement. However, for 15 iterations of decoding, interleaving doesn't give much improvement even when the interleaving depth is 15.

- Increasing the interleaving depth makes the symbol errors more random, but at the cost of a longer overall delay.

- We are currently investigating combining bit interleaving with symbol interleaving to give further improvement.

- **We can use the simulation results to estimate the actual performance:**

$$p_b < \frac{128}{255} \sum_{j=9}^{255} \frac{n_j}{N} \cdot \frac{j+8}{255} \qquad (4)$$

where

n_j is the # of blocks with j symbol errors

N is the total # of blocks

For the results to be accurate, N must be large.

- **Example: at SNR=1.21 dB, 5 iterations, interleaving depth =15,**

$$p_b < \frac{128}{255} \cdot \frac{23\times17+15\times18+6\times19+1\times20+1\times21+1\times23+2\times24}{21000\times255} \approx 8\times10^{-5}$$

compared to $p_b \approx 4\times10^{-9}$ assuming ideal interleaving (see previous curve).

- **When using symbol interleaving, the overall delay increases with interleaving depth. On the other hand, the performance of Turbo codes improves with the block size. As far as the delay is concerned, (symbol) interleaving to depth 5 is equivalent to one Turbo code block (block size=10200) containing 5 RS code words. The scheme with a larger Turbo code block is much better than the scheme with symbol interleaving.**

# of RS blocks = 21000 (# of Turbo blocks = 4200), SNR =1.21 dB, 15 iterations (no symbol interleaving between the RS codes in one Turbo block)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
20903	14	78	1	3	1	0	0	0	0	0	0	0	0	0

# of RS blocks with more than 8 symbol errors = 0

Using Feedback from the Outer Code

- **The performance of the concatenated scheme can be further improved by using feedback from the RS decoder to the Turbo decoder. With the help of feedback from the outer code, we can move the performance curve closer to capacity or get similar performance with fewer Turbo decoding iterations.**

- **Different rate RS codes can be used to provide feedback. After some iterations of Turbo decoding, the stronger RS codes are decoded and those bits are fed back to assist subsequent iterations of Turbo decoding; after some further iterations of Turbo decoding, the same process is repeated for the weaker RS codes.**

- **There are two different ways to use the feedback information in Turbo decoding:**

--The initialization of the extrinsic information L is changed. Normally this value is 0 before the first iteration, but based on the RS decoder output, the extrinsic information can be set to a predetermined value or changed by a predetermined amount.

--The channel metric of the information sequence is changed based on the RS decoder output.

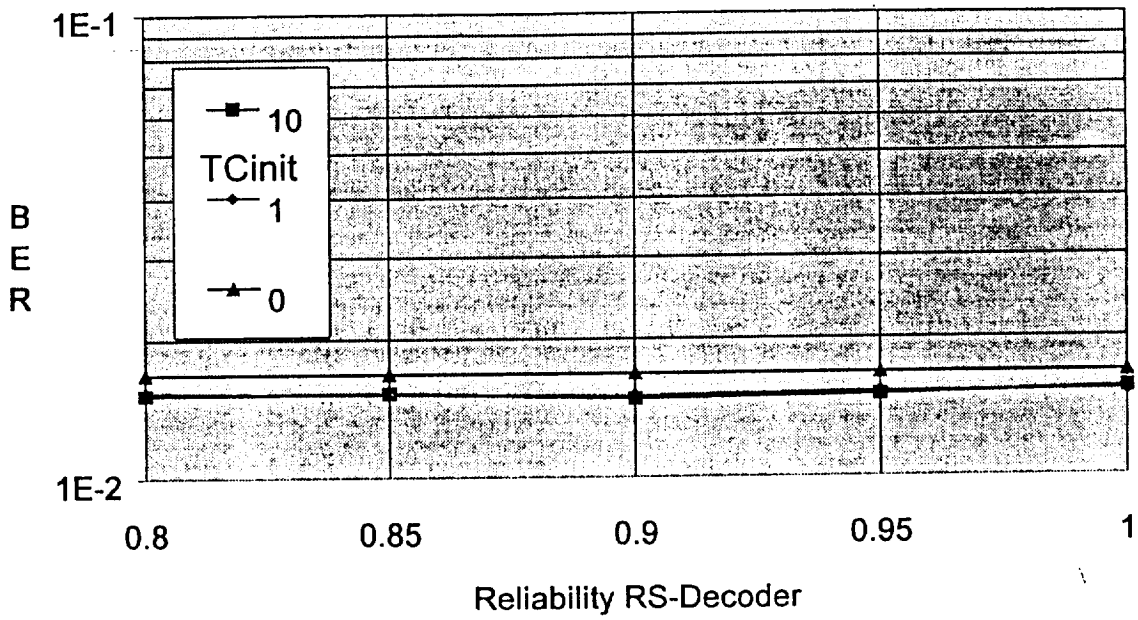
- **To see the different effect of the two methods, we assume that 10% of the information bits are fed back with a certain reliability (rate 1/2, block size 2040, Turbo code). The extrinsic information is initialized for the 'known' information bits with**

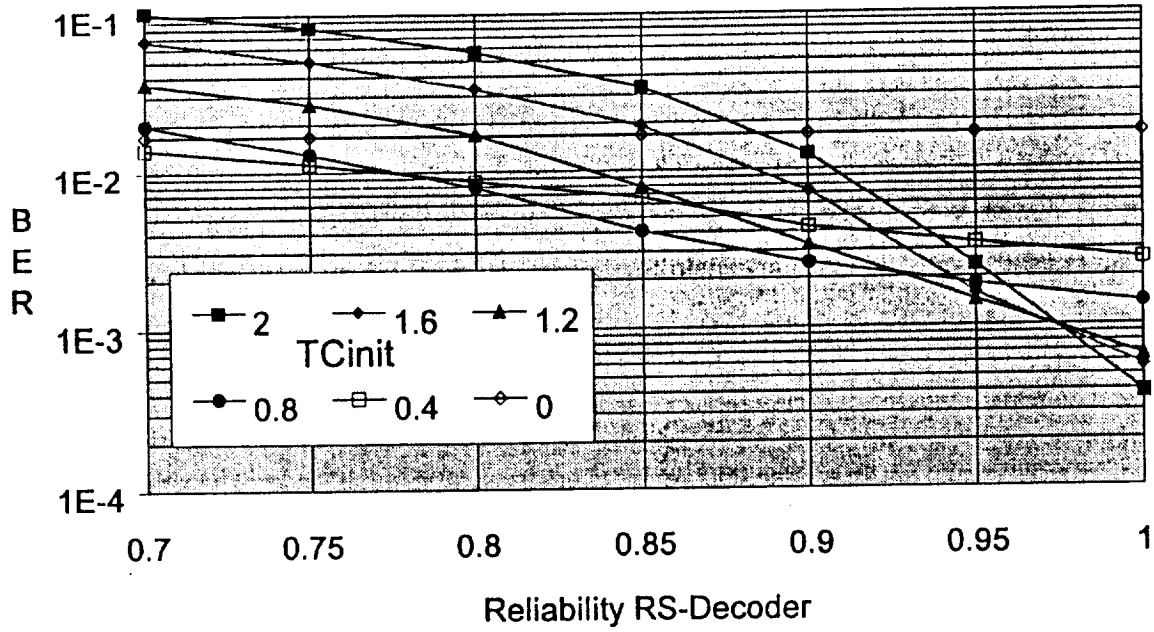
$$L = \pm TC_{init} \qquad (5)$$

or the channel metric of the information sequence is changed as follows:

$$\tilde{y}_t^{\,s} = y_t^{\,s} \pm TC_{init}, \qquad (6)$$

where $\pm$ depends on the estimated bits from the RS decoder.


1E-1
1E-2
BER
10
TCinit
1
0
0.8
0.85
0.9
0.95
1
Reliability RS-Decoder


1E-1
1E-2
1E-3
1E-4
BER
2
1.6
1.2
TCinit
0.8
0.4
0
0.7
0.75
0.8
0.85
0.9
0.95
1
Reliability RS-Decoder

- **The improvement achieved by changing the initialization values of the extrinsic information is negligible because L tends to grow to large values during the iterative decoding process for most of the information bits and a change in the initial value of the extrinsic information has its influence only for the first few iterations.**

- **A modification of the metrics of the received information sequence, however, has a significant effect on the decoding performance because a change in the information sequence maintains its influence during each decoding iteration. The optimal value of TC_{init} is a function of the channel SNR and the reliability of RS decoding. To give positive results, the feedback bits must have a high reliability.**

- **Hence we are focusing on feedback through a change in the channel metric. Two schemes are proposed:**

--Only a small fraction of the information bits in a block are encoded using one low rate RS code, or several codes with different rates, interleaved into the information sequence prior to Turbo encoding.

--All the information bits in a block are encoded using one high rate RS code, or several codes with different rates, interleaved, and then Turbo encoded.

- **Difference between the two concatenation schemes:**

--For the first scheme, the rate loss can be very small. Its main purpose is to reduce the number of iterations of Turbo decoding, although some lowering of the error floor may also result. This scheme can be quite natural in applications where some bits in a block need extra protection.

--For the second scheme, the number of iterations of Turbo decoding can also be reduced. Furthermore, after Turbo decoding, the outer codes can pick up most of the residual errors and lower the error floor significantly. But the rate loss is higher.

- **When to Feed Back**

--If the RS code is decoded correctly, better performance can be achieved if the decoded bits are fed back earlier and with a higher confidence level; if there are errors in the RS decoding, the performance may be worse than without feedback. Therefore, it is not wise to perform the feedback until after a fixed number of Turbo decoding iterations.

--For the commonly used Berlekamp-Massey algorithm for RS decoding, when the degree of the error location polynomial is greater than t or the number of roots is not equal to the degree, errors are detected, and the probability of undetected error is very low. This fact can be used to determine when to begin the feedback.

- **Preliminary results show that, for the first scheme, with a block size of around 65,000 , at a channel SNR =0.7 dB, a factor of 4 improvement in BER was achieved with a rate loss of only 3%.**

- **We are currently investigating the performance of these schemes in an attempt to find the best concatenation strategy.**

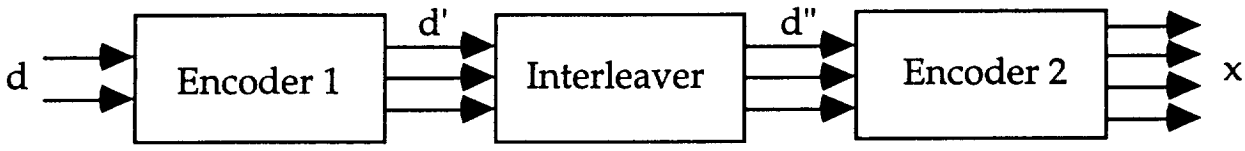
- **An outer BCH code can also be used to reduce average decoding complexity. With an outer 32 bit CRC, we can decrease the number of iterations. More than half of the blocks in a Turbo code with an interleaver size of 20400 bits can be decoded correctly with fewer than 4 iterations at an SNR= 0.7 dB.**

Part IV

Serial Concatenated Convolutional Codes

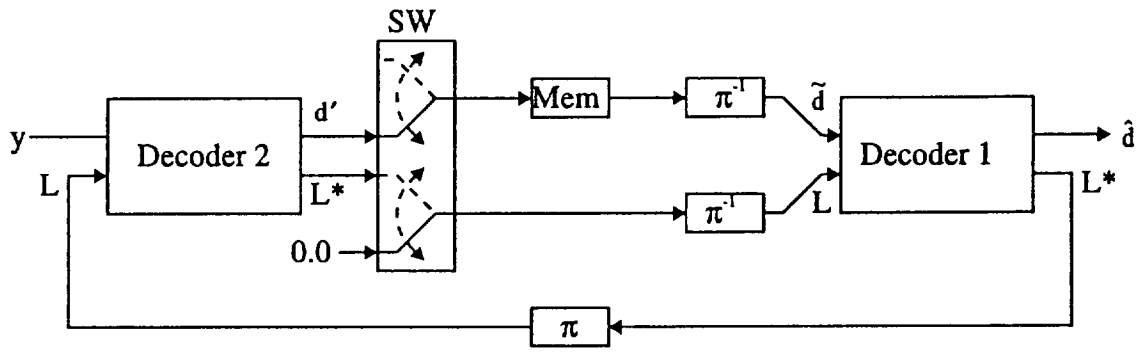
- **Turbo codes show that good performance can be achieved by using a parallel concatenation of convolutional codes, but it is also possible to get good performance using serial concatenation.**

- **Example of a rate 1/2 serial concatenated code:**

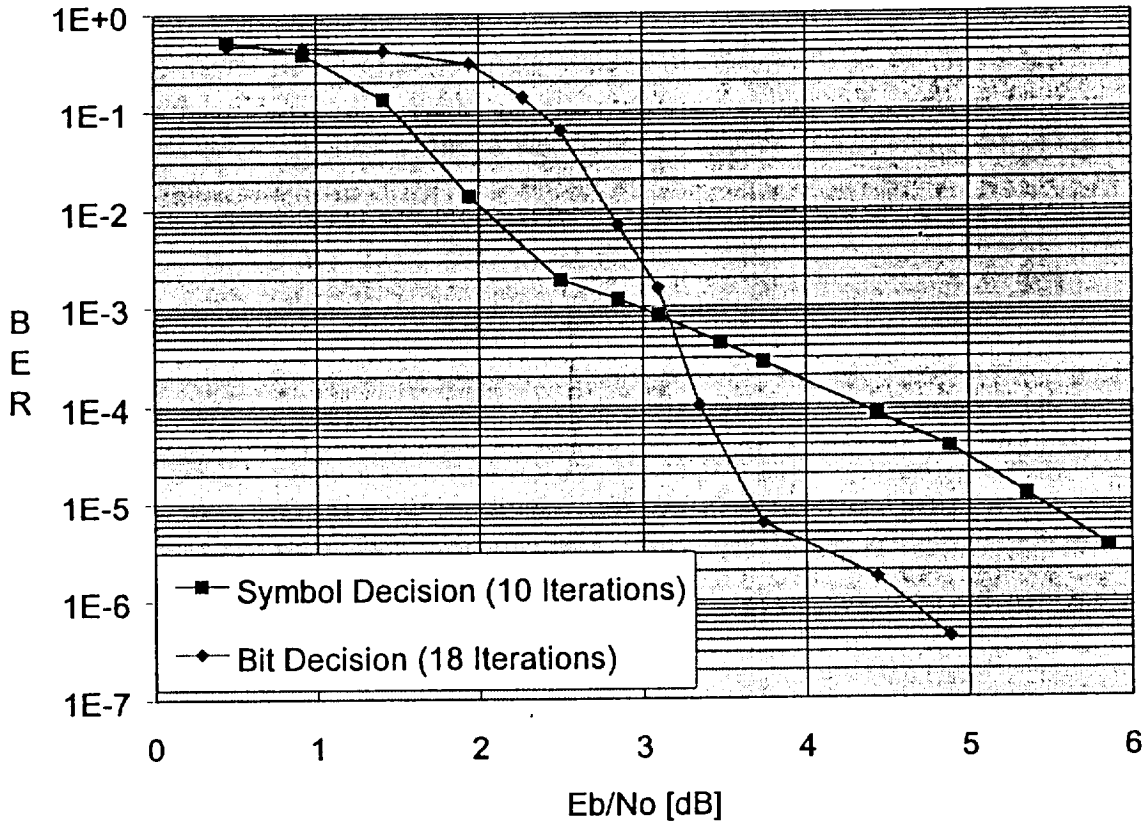


Encoder 1 is a rate 2/3 convolutional code, encoder 2 is a rate 3/4 convolutional code. The serial concatenated code has an overall rate of $R = \frac{2}{3} \cdot \frac{3}{4} = \frac{1}{2}$.

- **An iterative decoding process, which is similar to Turbo decoding, is proposed to decode the serial concatenated convolutional codes.**



- **Two decoding algorithms, one based on interleaving bits and the other based on interleaving symbols, have been tried.**



Comparison of different decoding algorithms for serial concatenated codes
(Block size: 400 bits, Rate: 1/2)

- **A comparison between the two decoding algorithms shows that: the symbol decisions algorithm is better for low SNR's; the bit decisions algorithm is better for high SNR's.**

- **Different component codes (that is, feedforward and feedback convolutional codes) were tried for the inner and outer codes. Results show that using a feedback inner code with a feedforward outer code gives better performance.**

- **Recently new progress with serial concatenation has been reported by Benedetto and Divsalar, which shows that this scheme can offer superior performance to Turbo codes, and code design rules are given.**

Topics for Further Research

- Consider the effect of interleaving only some of the sequences coming from the outer code instead of all of them.
- Code design rules for low SNR's.
- More than two (e.g., 3) component codes in the serial concatenation scheme.
- A thorough comparison of parallel concatenated convolutional codes and serial concatenated convolutional codes.