Two applications of a class of convolutional codes with reduced decoder complexity

th

e

Two applications of a class of convolutional codes with reduced decoder complexity

by

A.J. Vinck, A.C.M. Oerlemans and
T.G.J.A. Martens

E I N D H O V E N U N I V E R S I T Y O F T E C H N O L O G Y

Department of Electrical Engineering

Eindhoven The Netherlands

TWO APPLICATIONS OF A CLASS OF CONVOLUTIONAL CODES WITH REDUCED DECODER COMPLEXITY

by

A.J. Vinck
A.C.M. Oerlemans
T.G.J.A. Martens

TH-Report 80-E-115
ISBN 90-6144-115-3

Eindhoven
November 1980

Contents

Abstract

In this report we first discuss the implementation of a Fano decoder using the tree structure of the class $L_{2,\nu,\ell}$, of convolutional codes. Simulation results indicate that considerable reduction of the computational complexity can be obtained.
Secondly, we give a software as well as a hardware implementation for a restricted Viterbi decoder. This decoder has a complexity proportional to the number of estimates, m, stored in the decoder. For a representative value of the Binary Symmetric Channel (BSC) transition probability p_{BSC} of 0.045, the decoding bit error probability P_B decreases negative exponential with log(m). Hence, $P_B \approx c.2^{-\log(m)}$. Simulations indicate that low error probabilities can be obtained for small values of m, and utilization of the class $L_{2,\nu,\ell}$.

Vinck, A.J., A.C.M. Oerlemans and T.G.J.A. Martens
TWO APPLICATIONS OF A CLASS OF CONVOLUTIONAL CODES WITH REDUCED DECODER COMPLEXITY.
Department of Electrical Engineering, Eindhoven University of Technology, 1980.
TH-Report 80-E-115

Correspondence address:
Dr.ir. A.J. Vinck,
Group Information Theory,
Department of Electrical Engineering,
Eindhoven University of Technology,
P.O. Box 513,
5600 MB EINDHOVEN,
The Netherlands

I. Introduction

In communication systems, channel coding can be used to protect information against transmission errors. One of the major problems is to develop codes with low undetected error rates at a moderate complexity of the decoder. It is the aim of this report to introduce a class of codes that can be used to reduce decoder complexity.

In channel coding literature, one distinguishes between block and convolutional codes. A rate k/n block code can be generated with a combinatorial network. The n-output digits of this network at time t, only depend on the k-input digits at the same time instant. A convolutional code is generated with a sequential network. The n-output digits at time t depend on the k-input digits at time t, and on the input digits at time t-1,t-2,... . In the remainder of this report we assume that binary digits are to be transmitted over a binary symmetric channel (BSC) with a transition probability p_{BSC} and a rate 1/2 convolutional code. Fig. 1 gives a specific example of a

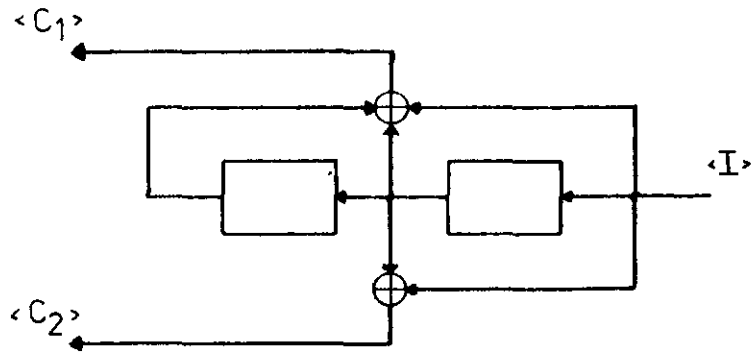


Fig. 1. A rate 1/2 convolutional encoder

binary rate 1/2 convolutional encoder. Formally, using the delay operator notation [1], the input/output relations can be written as

$$C_1(D) = I(D)g_1(D)$$

$$C_2(D) = I(D)g_2(D) \qquad (1)$$

where for the example of Fig. 1, $g_1(D) = 1+D+D^2$ and $g_2(D) = 1+D$, respectively. The number of memory elements, ν, of an encoder of the type as given in Fig. 1, is called the constraint length of the code. It indicates the maximum number of output pairs influenced by the memory of the encoder. From the shift register viewpoint, a convolutional code is the collection of all possible output streams of a particular encoder. A way of representing the encoder output as a function of its input and memory contents is by means of a tree. The tree associated with the encoder of Fig. 1, is given in Fig. 2.

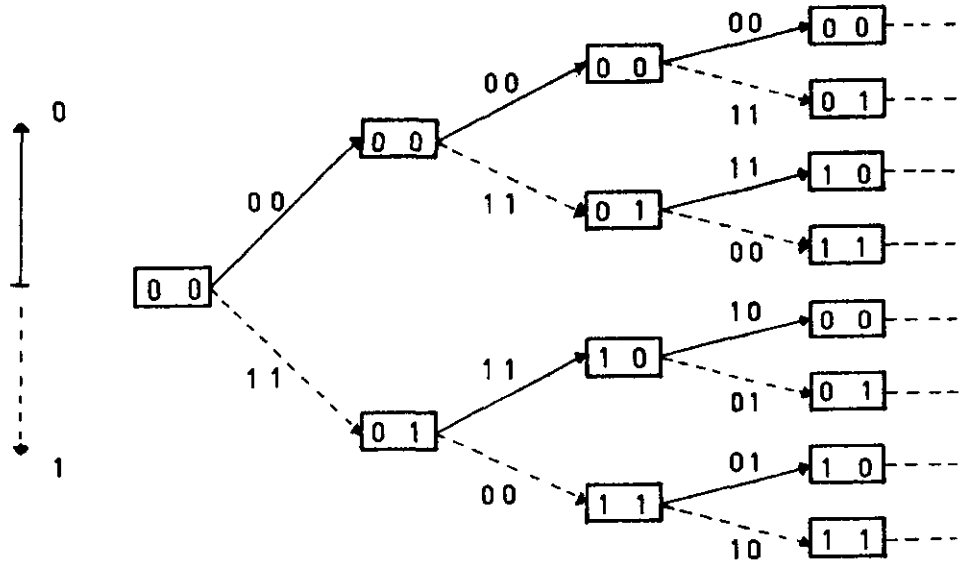


Fig. 2. Tree representation for the encoder of Fig. 1.

Each node in the tree is labeled with the information sequence leading to a particular node, or as the encoder has memory length, the last ν digits suffice as a label. Going from a node to a particular successor, the encoder outputs are placed along the branches, see Fig. 2. Note that the output of the encoder at a certain depth in the tree only depends on the node label and the present input. Hence, two nodes at a certain depth, with the same last ν digits of the respective labels, have the same encoded sequences following either node. Viterbi used this fact to develop a Maximum Likelihood (ML) decoder [2]. Note also, that the tree has $2^{**}t$ nodes at depth t. Hence, decoders that make use of the tree structure of convolutional codes are forced to do this in a clever way, in order to avoid exponential complexity growth. Fano [3], gave a solution to this problem for the class of random tree codes.

Let a binary information source generate independent digits with $Pr(0) = Pr(1) = 1/2$. The information is to be encoded with a rate 1/2 tree code. The binary code digits along the branches of the tree are also independently generated with $Pr(0) = Pr(1) = 1/2$. The probability that the encoder follows a path from the origin to some specific node at depth j is thus equal to $2^{**}(-j)$. Then, given a received sequence $\underline{y}$ of length 2j, define as a quality measure for the encoded path $\underline{c}$ to the node at level j

$$
\begin{aligned}
L_f(\underline{c}) &\triangleq \log Pr(\underline{c}/\underline{y}) \\
&= \log \frac{Pr(\underline{y}/\underline{c}) \cdot Pr(\underline{c})}{Pr(\underline{y})} \\
&= \log \frac{Pr(\underline{y}/\underline{c})}{Pr(\underline{y})} + \log Pr(\underline{c}) \qquad (2)
\end{aligned}
$$

Since our channel is memoryless, and the code is random, we have

$$L_f(\underline{c}) = \sum_{i=0}^{2j-1} \left[\frac{\log Pr(y_i/c_i)}{Pr(y_i)} \right] + \log 2^{-j}$$

$$= \sum_{i=0}^{2j-1} \left[\log Pr(y_i/c_i) + 1/2 \right] \qquad (3)$$

Massey [4], who ten years after Fano introduced this metric, shows that (3) is optimal. Thus, if we extend the node with greatest metric, and continue this process until we reach the end of the tree, we can be reasonable sure that we will have found the same path that an ML decoder would have found. Although the tree generated by a convolutional encoder is not random, it is random enough to use (3). Note that the metric as derived in (3) is additive. For example, when $P_{BSC} = 0.02$, one finds from (3), that

$$\log Pr(y_i/c_i) + 1/2 = \log(1.-0.02) + 1/2$$

$$= 0.47 \tilde{=} 0.5$$

when $y_i = c_i$, whereas

$$\log Pr(y_i/c_i) + 1/2 = \log(0.02) + 1/2$$

$$= -5.14 \tilde{=} -5.$$

when $y_i \neq c_i$. In practice, one scales the metrics so that all metric values can be closely approximated by integers. In this case, we scale with a factor of two. Note that if only paths of equal length are considered, the constant term 1/2, can be omitted. To illustrate the above described decoding strategy, consider the example of Fig.3.

The encoder connection polynomials are $1+D+D^2$ and $1+D^2$, respectively. The encoder output and corresponding Fano metrics are indicated along the branches. The increments in the Fano metric are easily calculated from the distance between a received pair and the branch transition pair. We have scaled the values of the respective Fano metrics to integer values.

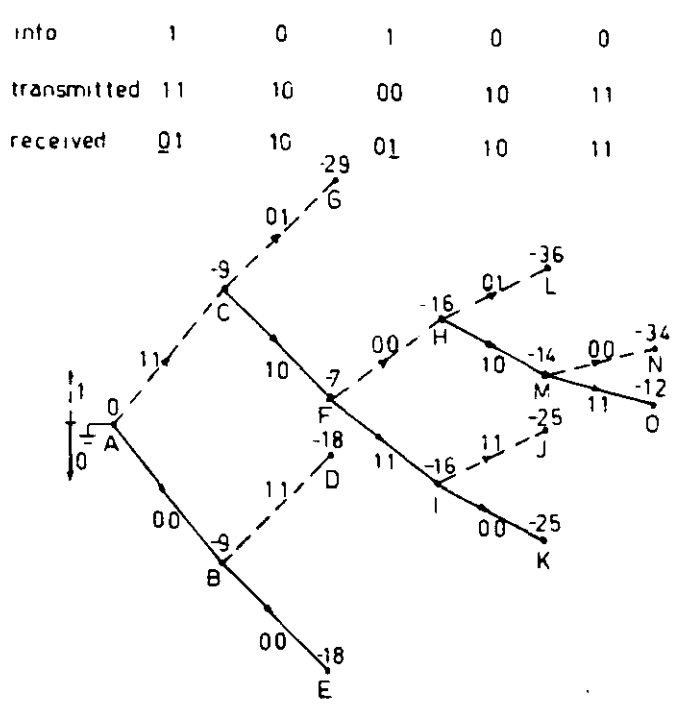


Fig. 3. Example of the operations of a tree decoder.

We now describe a class of convolutional codes with a special tree structure. This tree structure can be shown [5,6] to reduce the computational complexity of sequential decoding procedures. Let $g_1(D)$ and $g_2(D)$ represent the connection polynomials of a rate 1/2 convolutional encoder. This encoder is said to generate a code in the class $L_{2,\nu,\ell}$ of convolutional codes, iff

1). the constraint length of the encoder is equal to ν, and

2). the connection polynomials are pairwise equal for the first ℓ terms and unequal for the ℓ+1 th term. To end up with non delayed versions of other codes, both connection polynomials must have a constant term. As an example, the encoder of Fig. 1 generates a code in the class $L_{2,2,2}$. More concisely, a code is an element of $L_{2,\nu,\ell}$, iff

$$\text{max degree } [g_1(D),\ g_2(D)] = \nu \tag{4a}$$

$$\text{delay} \qquad [g_1(D) \oplus g_2(D)] = \ell \tag{4b}$$

$$\text{gcd} \qquad [g_1(D),\ g_2(D)] = 1 \tag{4c}$$

Condition (4c) means that both connection polynomials have no common divisor unequal to 1, or that the code is non catastrophic. That is, infinite degree nonzero information sequences cannot produce finite degree code sequences. Furthermore, if (4c) is fulfilled, an instantaneous inverse to the code sequence can be derived, using Euclid's [1] algorithm. The $\oplus$ sign means mod-2 addition of the polynomials.

At this moment we restrict ourselves to describe how the above defined class of codes influences the tree structure of the general class of binary rate 1/2 convolutional codes.

Following Massey [7], the information and corresponding code vector sequence from time u up to time v, are given as

$$i_{[u,v)} = (i_u, i_{u+1}, \ldots, i_{v-1}), \text{ and}$$

$$\underline{c}_{[u,v)} = (\underline{c}_u, \underline{c}_{u+1}, \ldots, \underline{c}_{v-1}),$$

respectively. For example, for the tree in Fig. 2 the information input sequence $i_{[0,2)} = (1,1)$, gives rise to a corresponding code vector sequence $\underline{c}_{[0,2)} = ((11), (00))$. Furthermore, we can split up the information sequence $i_{[u,v)}$ into a concatenation of substrings, as $i_{[0,v)} = i_{[o,u)} * i_{[u,v)}$. The same can be done for the code vector sequence $\underline{c}_{[o,v)}$. We are now ready to describe some important properties of the class $L_{2,\nu,\ell}$ of convolutional codes as given in (4).

Suppose we fully developed a code tree up to depth v. Then, this code tree can be divided into subtrees, by taking together all paths with the same path history up to time $(v-\ell)$, $0<\ell\leq\nu$. In the following lemma we proof that given an information and a corresponding code vector sequence from a particular subtree, we can find back or derive the whole subtree again.

Lemma 1: Given an information sequence, and hence the corresponding code vector sequence, up to time v, then $2^{\ell}-1$ other code vector sequences can be derived from the given one.

Proof: Let

$$i_{[o,j+\ell)} = i_{[0,j)} * i_{[j,j+\ell)}, \text{ and}$$

$$i^1_{[0,j+\ell)} = i_{[0,j)} * i^1_{[j,j+\ell)}$$

be two information sequences diverging at depth j in the tree. The two corresponding code vector sequences are

$$\underline{c}_{[0,j+\ell)} = \underline{c}_{[o,j)} * \underline{c}_{[j,j+\ell)}, \text{ and}$$

$$\underline{c}^1_{[0,j+\ell)} = \underline{c}_{[0,j)} * \underline{c}^1_{[j,j+\ell)},$$

respectively. As $i^1_{[0,j)} = i_{[o,j)}$, the code vector substrings $\underline{c}^1_{[0,j)}$ and $\underline{c}_{[0,j)}$ must also be equal. Hence, the two code vector sequences $\underline{c}_{[0,j+\ell)}$ and $\underline{c}^1_{[0,j+\ell)}$ only differ in the last ℓ stages. As the first ℓ coefficients of $g_1(D)$ and $g_2(D)$ of a code in $L_{2,\nu,\ell}$ are equal, the differences, $\underline{c}_{[j+i,j+i+1)} \oplus \underline{c}^1_{[j+i,j+i+1)}$, are elements of $\{(00),(11)\}$ for all i, $0 \leq i \leq \ell-1$.

Property 1: Due to the structure of the class $L_{2,\nu,\ell}$, we can find an information sequence $i^{\Delta}_{[0,j+\ell)}$, that leads to a code vector sequence $\underline{c}^{\Delta}_{[0,j+\ell)}$ with the property that

$$\underline{c}^{\Delta}_{[i,i+1)} = \begin{cases} (0\ 0) & i \neq j, 0 \leq i \leq j+\ell-1 \\ (1\ 1) & i = j \end{cases}$$

The first nonzero component of $i^{\Delta}_{[0,j+\ell)}$ is the component $i^{\Delta}_{[j,j+1)}$. For the tree of Fig. 2, $i^{\Delta}_{[0,j+\ell)} = i^{\Delta}_{[0,j+2)} = \emptyset_{[0,j)} * (1,1)$, where $\emptyset_{[0,j)}$ is the all zero sequence up to time j.

Let $i_{[0,j+\ell)}$ be some information sequence representing a path up to depth $(j+\ell)$ in the tree. From Lemma 1 we saw that we can derive

$2^{\ell}-1$ other information sequences and code vector sequences from this path. Pick the special path $i^{1}_{[0,j+\ell)} = i_{[0,j+\ell)} \oplus i^{\Delta}_{[0,j+\ell)}$, and compare the respective code vector sequences. According to property 1, $\underline{c}_{[0,j+\ell)}$ and $\underline{c}^{1}_{[0,j+\ell)}$ only differ at depth j, with a component equal to (1 1). The remaining components of the respective code vector sequences are equal. Thus, if according to Lemma 1, $i_{[0,j+\ell)}$ represents a class of 2^{ℓ} paths, then, this class can be split up into two subclasses of $2^{\ell-1}$ paths. One subclass can be represented by the path $i_{[0,j+\ell)}$, whereas the other may be represented by a path $i^{1}_{[0,j+\ell)} = i_{[0,j+\ell)} \oplus i^{\Delta}_{[0,j+\ell)}$. Note that if the information estimate $i_{[0,j+\ell)}$ is the best estimate within its subclass, then $i^{1}_{[0,j+\ell)}$ is also the best estimate within its subclass. If we define the metric $L_f[\underline{c}_{[0,j+\ell)}]$ to equal the Fano metric corresponding with an information sequence estimate $i_{[0,j+\ell)}$, then the Fano metric $L_f[\underline{c}^{1}_{[0,j+\ell)}]$ corresponding with the estimate $i^{1}_{[0,j+\ell)} = i_{[0,j+\ell)} \oplus i^{\Delta}_{[0,j+\ell)}$ follows easily from the metric $L_f[\underline{c}_{[0,j+\ell)}]$. To wit

$$L_f\,[\underline{c}^{1}_{[0,j+\ell)}] = L_f[\underline{c}_{[0,j+\ell)}] + d[(1\ 1) : \underline{n}_{[j,j+1)}],$$

where

$$d[(1\ 1) : \underline{n}_{[j,j+1)}] = \begin{cases} 2\log(1-p)/p\ , & \text{if } \underline{n}_{[j,j+1)} = (0\ 0) \\ -2\log(1-p)/p\ , & \text{if } \underline{n}_{[j,j+1)} = (1\ 1) \\ 0 & \text{otherwise} \end{cases} \qquad (5)$$

This is an important property for the decoding algorithms to be described. We will now show how the above can be used in a decoding algorithm that uses the Fano metric.

Let $i_{[0,j+1)}$ be the common part of a subclass of $2^{\ell-1}$ sequences represented by the best, or one of the best, estimate $i_{[0,j+\ell)}$. We extend the representative sequence $i_{[0,j+\ell)}$ to $i_{[0,j+\ell+1)} = i_{[0,j+\ell)} * i_{[j+\ell,j+\ell+1)}$ such that $\underline{n}_{[j+\ell,j+\ell+1)} = \underline{c}_{[j+\ell,j+\ell+1)} \oplus \underline{r}_{[j+\ell,j+\ell+1)} \varepsilon ((0\ 0), (1\ 0))$. As both connection polynomials have a constant term, this is always possible. Because of (4b), all other possible extensions of sequences of this particular subclass give rise to a difference of (0 0), or (1 1) with $\underline{n}_{[j+\ell,j+\ell+1)}$. Hence, if $i_{[0,j+\ell)}$ is the best estimate within the subclass of $2^{\ell-1}$ paths, the extension $i_{[0,j+\ell+1)}$ is the best estimate within a class of 2^{ℓ} paths. Fig. 4 gives a flowchart of the above method.

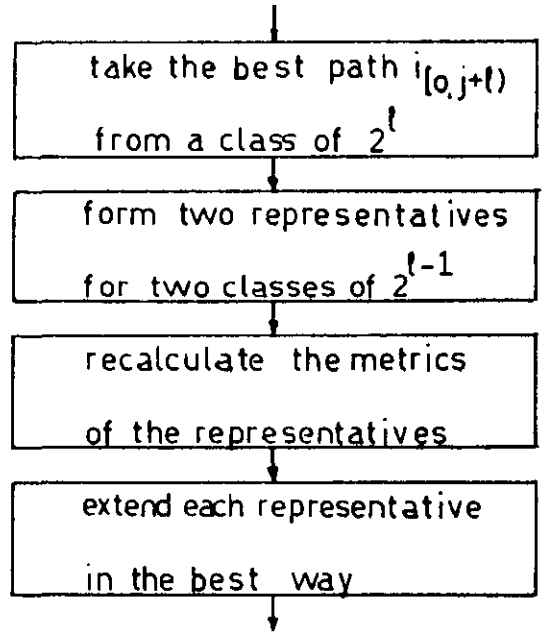


Fig. 4. Flowchart of the extension of a class of 2^{ℓ} paths.

One of the important things about coding is the undetected decoding error rate. For convolutional codes, the minimum Hamming distance between any two code vector sequences to a large extent determines the undetected error rate. This minimum Hamming distance is called the free distance, or d_{free}. Table I shows some values of d_{free} for several codes in the class $L_{2,\nu,\nu-1}$. This class of codes can be seen as a mirrored version of the QLI [8] codes, that permit easy data recovery from the received data stream. This is also possible for codes in $L_{2,\nu,\nu-1}$.

Table I

List of d_{free} for some codes in $L_{2,\nu,\nu-1}$.

ν	Type	g_1	g_2	g_1^{-1}	g_2^{-1}	d_{free}
2	$L_{2,2,1}$	5	7	3	2	5
4	$L_{2,4,3}$	31	33	10	17	7
6	$L_{2,6,5}$	113	153	75	52	9
11	$L_{2,11,10}$	4253	6253	2264	3403	12
12	$L_{2,12,11}$	14253	10253	7415	4272	13
15	$L_{2,15,14}$	104253	144253	66166	45701	14
16	$L_{2,16,15}$	204253	304253	127333	144554	14
22	$L_{2,22,21}$	23604253	33604253	16556734	13535153	19
31	$L_{2,31,30}$	32642356253	22642356253	16274741142	11520660725	23

The code generators g_1 and g_2 are given in octal notation. In sequential decoding, the most important parameter is the column distance function [9]. It is defined as

$$d_c(r) \triangleq \min_{d_o^{ij} \neq 0} \sum_{s=0}^{r-1} d_s^{ij}$$

where d_s^{ij} denotes the Hamming distance between the s branches of two encoder output vector sequences. A large value of $d_c(r)$ for values between 0 and 15 garantees a good distribution in the number of computations for a decoding job to be performed. Johannesson [9] shows that rapid column distance growth minimizes the decoding effort, and therefore the probability of decoding failure. For the ν=31 code in Table I, the column distance grows as 2^{30},3,4,5,5,6,7,7,8,8,9,9, 10,10,11,11. Johannesson [9] also gives codes with a rapid column distance growth. For instance for the code with encoder polynomials g_1 =103745, and g_2=164133 the column distance grows as 2,3,3,4,4,5,5, 6,6,6,7,7,... . This distance growth is much different from that of the codes in Table I, and hence, it is somewhat surprising that they yield such good performance in conjunction with sequential decoding algorithms. This is the subject of the next chapters.

The last part of this introduction is to show how a decoder can make use of the received data vector sequence in order to give an estimate of the encoded information sequence. If we omit the delay operator notation, the input/output relation of the encoder can be written as

$$\underline{c} = (c_1, c_2)$$
$$= IG$$
$$= (Ig_1, Ig_2).$$

After transmission over a BSC the code vector sequence $\underline{c}$ is received as

$$\underline{r} = (r_1, r_2) = \underline{c} \oplus \underline{n},$$
$$= (Ig_1 \oplus n_1, Ig_2 \oplus n_2),$$

As g_1 and g_2 have no common factor, we can find a matrix $G^{-1} = (g_1^{-1}, g_2^{-1})$ such that $g_1 g_1^{-1} \oplus g_2 g_2^{-1} = 1$. Then, the inverse to the received data vector sequence is defined as

$$I \oplus e = \underline{r}(g_1^{-1}, g_2^{-1})^T$$
$$= I \oplus n_1 g_1^{-1} \oplus n_2 g_2^{-1}.$$

We define the syndrome sequence z as

$$z = \underline{r}(g_2, g_1)^T$$
$$= n_1 g_2 \oplus n_2 g_1.$$

The task of the convolutional decoder is to find an error vector sequence $\underline{n}$, according to some metric criterion, that may be a possible cause of the syndrome sequence z. The inverse to this estimate is defined as

$$\hat{e} \triangleq \hat{n}_1 g_1^{-1} \oplus \hat{n}_2 g_2^{-1},$$

and must be added to (I ⊕ e), to give an estimate of the information sequence. Another possibility is to compare a possible code sequence IG, with the received data stream $\underline{r}$. As

$$(n_1,n_2) = (n_1,n_2)\begin{pmatrix} g_1^{-1} & g_2 \\ g_2^{-1} & g_1 \end{pmatrix}\begin{pmatrix} g_1 & g_2 \\ g_2^{-1} & g_1^{-1} \end{pmatrix}$$

$$\triangleq (e,z)\begin{pmatrix} g_1 & g_2 \\ g_2^{-1} & g_1^{-1} \end{pmatrix}$$

one could also compare a possible code vector sequence $\hat{e}G$ with the vector sequence $z(g_2^{-1},g_1^{-1})$, in order to find the sequence $\hat{e}$ that leads to the information sequence $\hat{I}$ directly.

For an additive white Gaussian noise channel (AWGN) with hard quantization at the matched filter output, the previously mentioned encoding and syndrome forming circuits, together with the invertor G^{-1}, are used as indicated in Fig. 5.

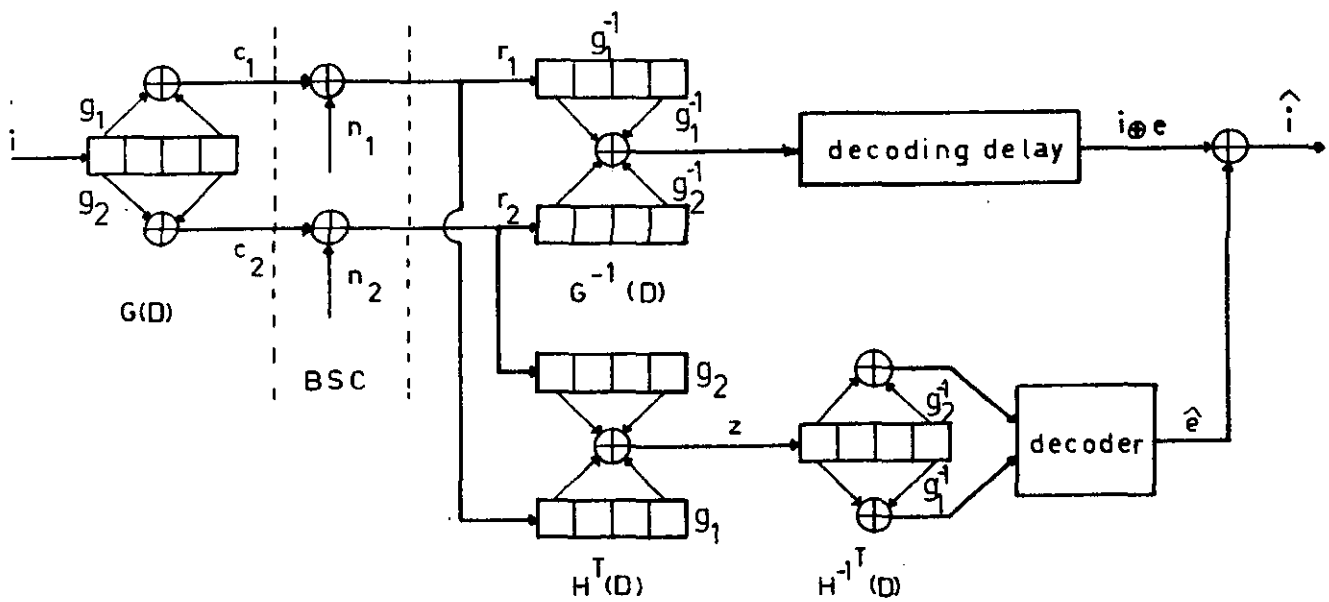


Fig. 5. Schematic use of a convolutional code/decoder

II. Fano decoding

In this chapter we first shortly explain the operations of the Fano decoding algorithm. Then, we investigate the influence of the class $L_{2,\nu,\ell}$ of convolutional codes on the decoder complexity.

Fano decoding is a method to decode convolutional codes sequentially. That is, a search along a possible path through the code tree is done one branch at a time. If we connect the Fano metric with each node in the tree, the decoder may move forward from a particular node to a successor node, iff the Fano metric T for the successor node exceeds or equals a certain threshold T_0. If the decoder cannot move forward without violating the threshold T_0, it has to return to a preceding node, in order to try an alternate path. If the backward move is also impossible, the decoder lowers the threshold T_0 with a fixed value τ. If the threshold is lowered, then the decoder tries to move forward again. The threshold T_0 is forced to move upward in discrete steps of size τ to be as close to T as possible, iff a node is visited for the first time. This is to avoid looping of the algorithm. The decoder continues in the above way until the end of the tree is reached. The case where the tree has both a starting and an end point is referred to as frame decoding. In frame decoding the last encoded information digit is followed by ν all zero digits, thus resetting the encoder in the all zero state. As the Fano decoder can give a decision before searching through all possible paths, the algorithm is suboptimal with respect to the decoding error probability of an ML decoder. However,

the loss is very small. The main advantage of the Fano decoder is the small memory necessity of the decoder, while the number of investigated paths is small compared to the number of paths an ML decoder investigates.

The description of the Fano decoding algorithm is given in comprehensive form in Jelinek [10]. We only give the flowchart, see Fig. 6, where the value of $T[i_{[0,j)}]$ is defined as

$$T[i_{[0,j)}] \triangleq \left\lfloor L_f[\underline{c}_{[0,j)}] \right\rfloor ,$$

and the values of the label c_j^* and the inverse labelling function $s(c_j^*)$ can be found in Tables II and III, respectively.

TABLE II
Values for c_j^*.

	$i_{[j,j+1)} = 0$	$i_{[j,j+1)} = 1$
$T(i_{[0,j)} * 0) \geq T(i_{[0,j)} * 1)$	0	1
$T(i_{[0,j)} * 0) < T(i_{[0,j)} * 1)$	1	0

TABLE III
Values for $s(c_j^*)$.

	$c_j^* = 0$	$c_j^* = 1$
$T(i_{[0,j)} * 0) \geq T(i_{[0,j)} * 1)$	0	1
$T(i_{[0,j)} * 0) < T(i_{[0,j)} * 1)$	1	0

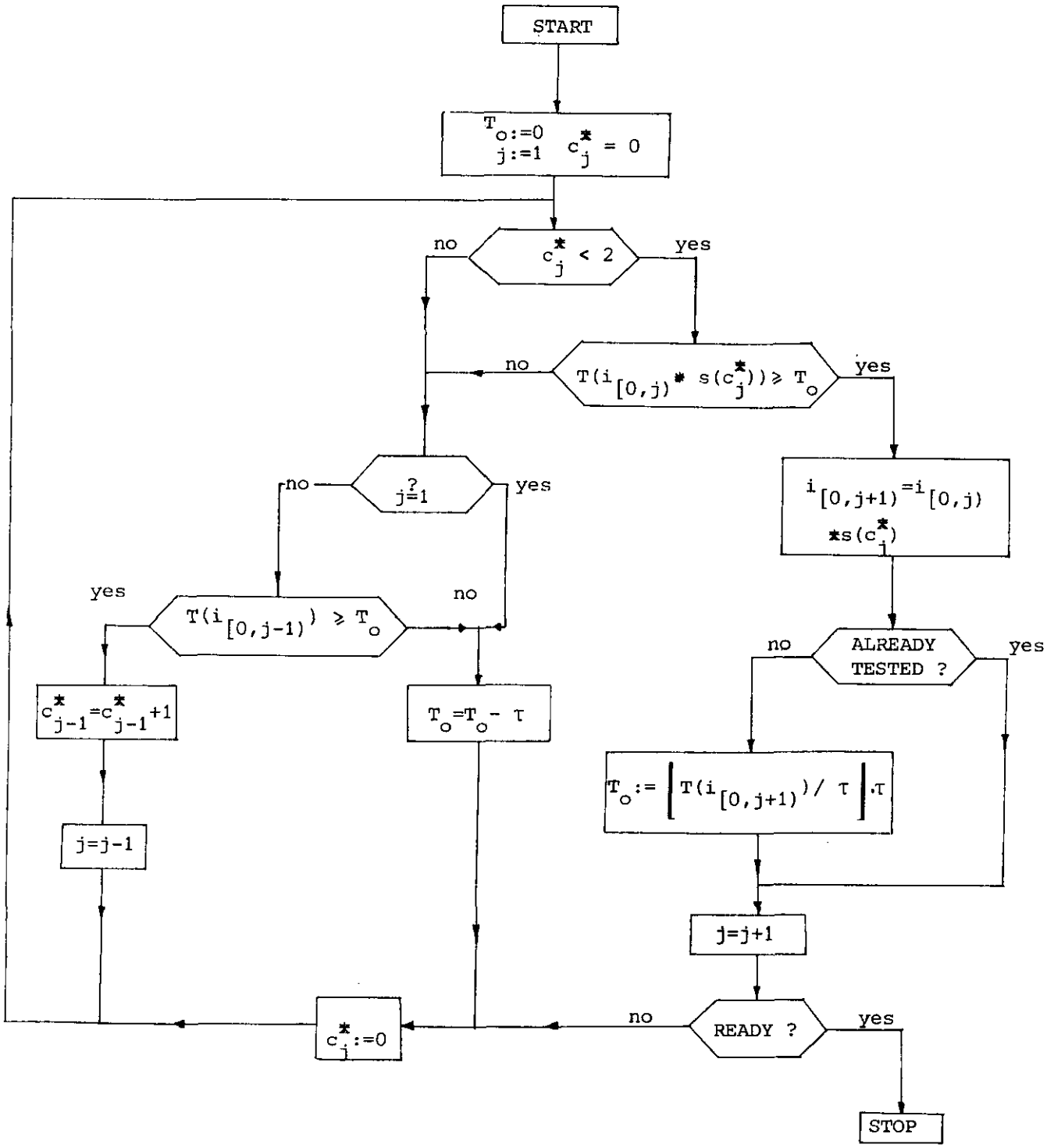


Fig. 6. Flowchart of the Fano decoding algorithm.

Encoder G = $[1+D+D^2, +D^2]$

Information	1	0	1	0	0
Transmitted	11	10	00	10	11
Received	01	10	01	10	11

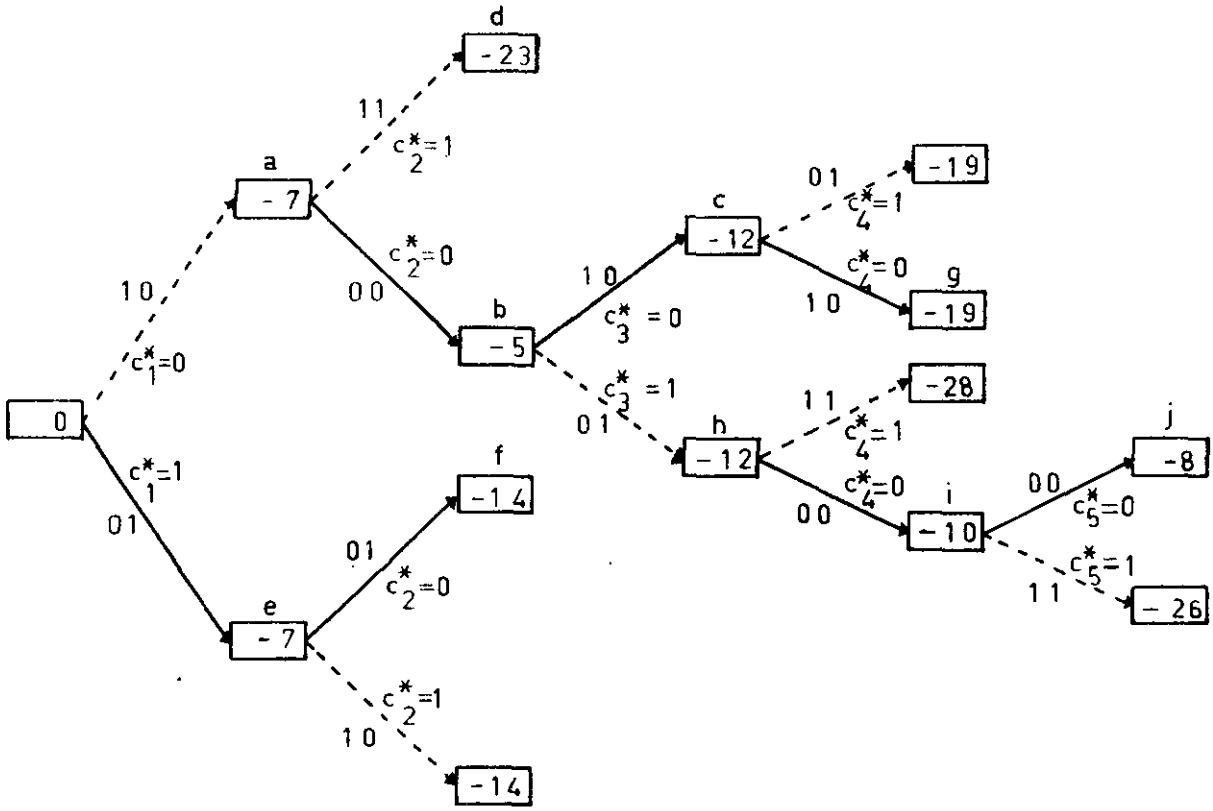


Fig. 7. Fano metrics for a specific received sequence.

Table IV traces the steps of the Fano decoder in decoding a specific data vector sequence. The example is taken from [11]. In Fig. 7 we represent the various values of the Fano metrics in a tree, and we indicated along the branches the value of the estimated noise, followed by the branch labels, respectively. A dashed line indicates that a path leading to a specific node is extended

with an information digit equal to zero.

TABLE IV

Decoder actions for tree of Fig. 7.

location	action	violate T_o	action	violate T_o	action	T_o	j	c^*_j	arrow
root	look at a	YES			decrement T_o	-4	1	0	B
root	look at a	YES			decrement T_o	-8	1	0	B
root	look at a	NO			go to a	-8	2	0	D
a	look at b	NO			go to b	-8	3	0	C
b	look at c	YES	look at a	NO	go to a	-8	1	1	A
a	look at d	YES	look at root	NO	go to root	-8	2	1	A
root	look at e	NO			go to e	-8	2	0	C
e	look at f	YES	look at root	NO	go to root	-8	1	2	A
root	$c^*_1 = 2$				decrement T_o	-12	1	0	B
root	look at a	NO			go to a	-12	2	0	D
a	look at b	NO			go to b	-12	3	0	D
b	look at c	NO			go to c	-12	4	0	D
c	look at g	YES	look at b	NO	go to b	-12	3	1	A
b	look at h	NO			go to h	-12	4	0	C
h	look at i	NO			go to i	-12	5	0	C
i	look at j	NO			go to j	-8	6	0	C

As was explained before, given a received vector sequence $\underline{y}$, the Fano decoding algorithm is a search along a possible path through the code tree, one branch at a time. If we use the tree structure of the class $L_{2,\nu,\ell}$, then, from each node in the tree we can derive a class of $2^{**}\ell$ possible paths. Now, let us represent each node in the tree by the node that follows from the best possible path within its class. Then, this tree can be seen as a contraction of the original tree. The extension of a node can be summarized as follows.

$$i_{[0,j+\ell)} \longmapsto \begin{cases} i_{[0,j+\ell+1)} & \text{if } i_{[j+\ell,j+\ell+1)} \\ \underline{n}_{[j+\ell,j+\ell+1)} \in \{(00),(01)\} & \end{cases},$$

$$i_{[0,j+\ell)} \overset{\Delta}{*} i_{[j+\ell,j+\ell+1)}$$

$$i_{[0,j+\ell)} \longmapsto \begin{cases} i^1_{[0,j+\ell+1)} & \\ \underline{n}^1_{[j+\ell,j+\ell+1)} \in \{(00),(01)\} & \end{cases},$$

and

$$L_f[\underline{c}_{[0,j+\ell+1)}] = \begin{cases} L_f[\underline{c}_{[0,j+\ell)}]+2\log(1-p)+1 & \text{if } \underline{n} = (00) \\ L_f[\underline{c}_{[0,j+\ell)}]+\log p(1-p)+1 & \text{if } \underline{n} = (01) \end{cases} \tag{6a}$$

$$L_f[\underline{c}^1_{[0,j+\ell+1)}] = \begin{cases} L_f[\underline{c}_{[0,j+\ell+1)}]+2\log(1-p)/p & \text{if } \underline{n}_{[j,j+1)} = (00) \\ L_f[\underline{c}_{[0,j+\ell+1)}] & \text{otherwise} \end{cases} \tag{6b}$$

In Table V and Fig.8 we trace the actions of the Fano decoder for the example of Fig. 7 for the modified Fano decoder.

TABLE V

Decoder actions corresponding to Fig. 8.

location	action	violate T_o	action	violate T_o	action	T_o	j^*	$c_{j^*}^*$
root	look at a	YES	decrement T_o			-4	1	0
root	look at a	YES	decrement T_o			-8	1	0
root	look at a	NO	go to a			-8	2	0
a	look at b	NO	go to b			-8	3	0
b	look at c	YES	look at a	NO	go to a	-8	2	1
a	look at d	YES	look at root	NO	go to root	-8	1	1
root	look at e	YES	decrement T_o			-12	1	0
root	look at a	NO	go to a			-12	2	0
a	look at b	NO	go to b			-12	3	0
b	look at c	NO	go to c			-12	4	0
c	look at f	NO	go to f			-12	5	0
f	look at g	NO	go to g			-8	6	0

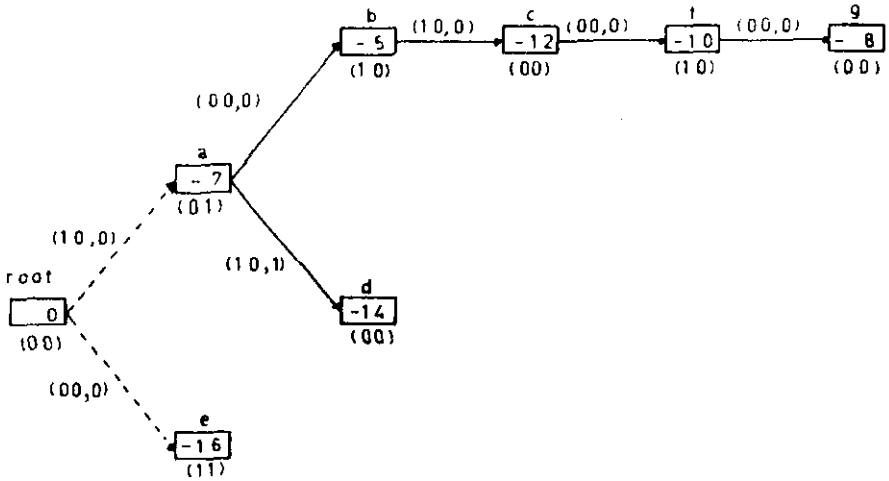


Fig. 8. Path followed by the modified Fano decoder for the example of Fig. 7.

We have indicated the last two digits leading to the respective nodes. Between parentheses are the estimated noise digits, followed by c_j^*.

According to (6b), a negative contribution is added to the Fano metric of a representative path, if $\underline{n}^1_{[j,j+1)}=(11)$. This, however, is done ℓ time units later than the estimated noise occurs. This phenomenon might cause additional decoding errors. For, there could be a path with higher overall threshold function, see Fig. 9.

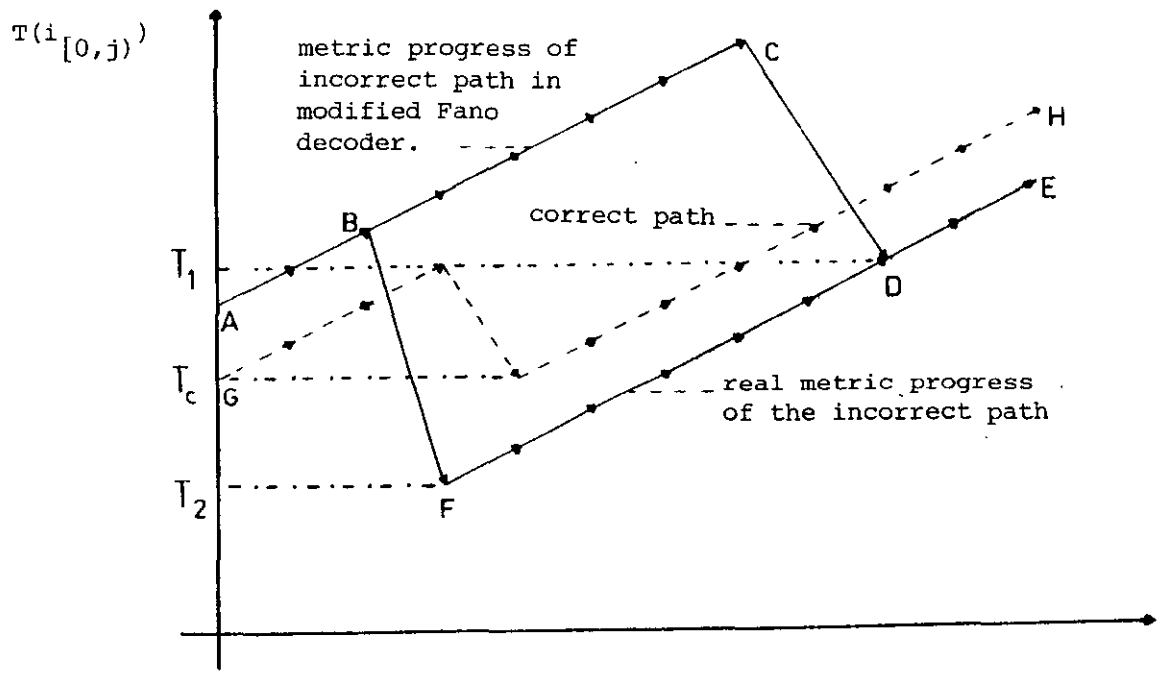


Fig. 9. Metric progress for incorrect path BCD, and BFD in the classical and the modified Fano decoder, respectively.

No additional errors are made if the incorrect path remains unmerged with the correct one, like for pure tree codes. As convolutional codes are linear trellis codes, paths can merge. The influence on the decoding error probability will be discussed in Chapter III.

III. Simulations for the Fano decoding algorithm

Simulations were carried out for the Fano decoder using the class $L_{2,\nu,\ell}$, and for ODP codes without using symmetries. For ODP codes the first $(\nu+1)$ terms of the column distance function are best. We divided the information into frames of 256 digits followed by ν all zero digits. These $(256+\nu)$ digits were encoded and transmitted over a BSC with transition probability of 0.033, 0.045 and 0.057, respectively. A particular simulation run consists of 25.000 frames each. The decoding delay was set equal to the frame length, unless stated otherwise. In Table VI, we give the contributions for the Fano metric for various noise pair estimates and

Table VI. Metric contributions.

$\underline{\underline{n}}[j,j+1)$ \ p_{BSC}	0.033	0.045	0.057
0 0	1	2	1
0 1	-4	-7	-3
1 0	-4	-7	-3
1 1	-9	-16	-7

channel transition probabilities. The values of the threshold step size τ are given in the figures.

In Fig. 10 we plot the distribution of the normalized number of forward backward and nonsteps, numbered as 2,3,4, respectively. The total number of steps is numbered 1. As can be seen, the non steps are of minor importance for the complexity of the decoder. Therefore, we define a computation to be a forward or a backward move of the decoder.

The ODP [9] is an important parameter in sequential decoding. Hence, our initial simulation runs apply to ODP codes. The results are given in Figs. 10 through 16. Observe that the distribution of the number of computations does not change appreciable if we increase ν beyond the value ν=15. The dependance of the results for smaller values of ν is a result of the trellis structure of convolutional codes. If we neglect the influence on the computational distribution, then the advantage that accrues from using long constaint length codes is an improvement in the undetected error rate. In the same figures one can also note the influence of the decoding length. The influence of the threshold step size τ, see Fig. 15, is given for one specific ν=15 ODP code. Figs. 17,18 and 19 summarize the results for the class $L_{2,\nu,\ell}$, using the tree structure discussed previously. For these codes, we optimized the column distance function. At p_{BSC}=0.045, we compare some $L_{2,\nu,\nu-1}$ codes with the ν=23 ODP code, see Fig. 20. In order to see the influence of the tree structure of the class $L_{2,\nu,\nu-1}$, we simulated a code from $L_{2,\nu,\nu-1}$ with optimized column distance function, see Fig. 21, under various circumstances. Line 1 gives the performance when these codes are used without using the tree structure. Line 2 are the simulation results when only classes with equal metrics are

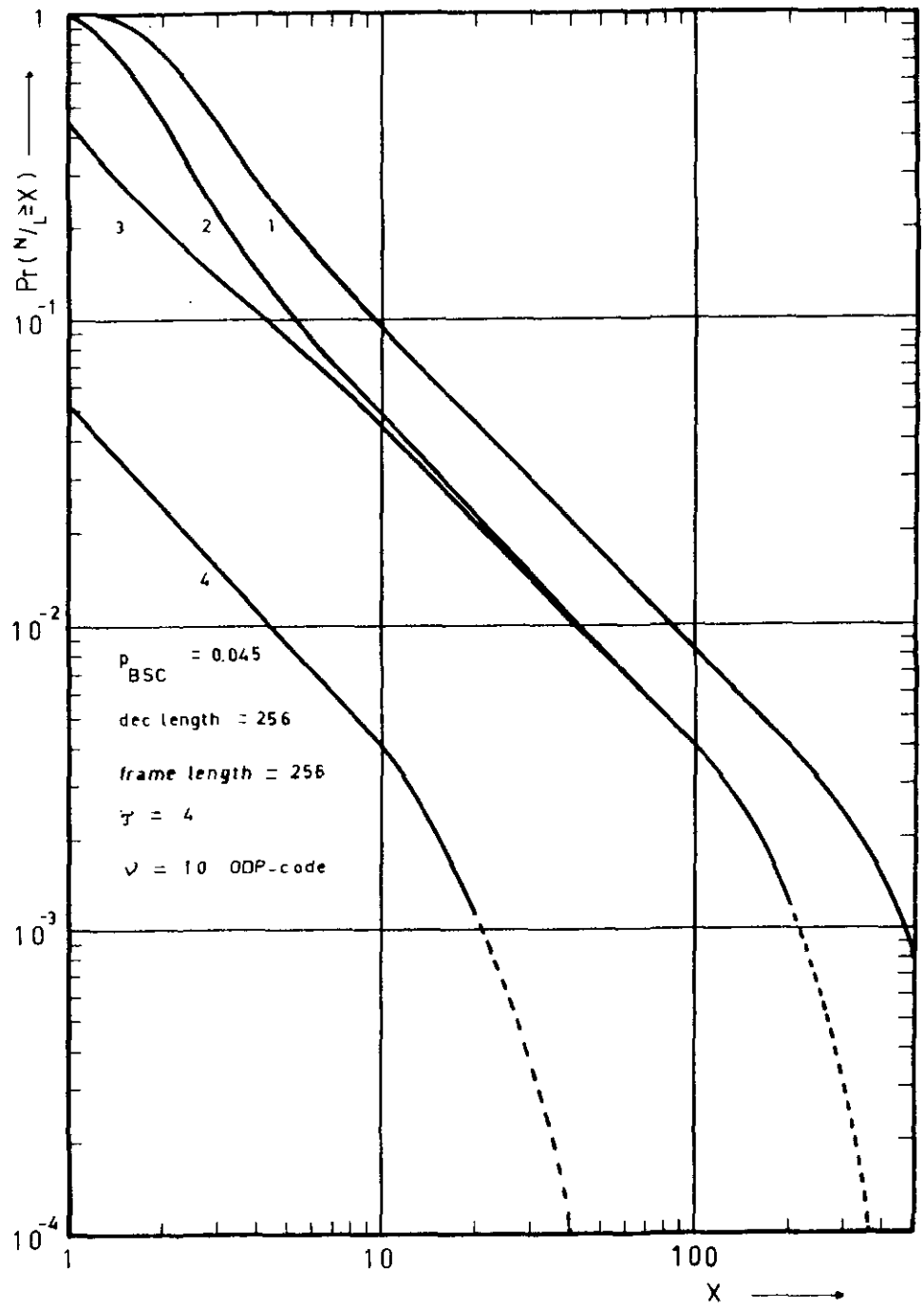


Fig. 10 Total-, forward-, backward- and non-step computational distribution for ν=10 ODP code.

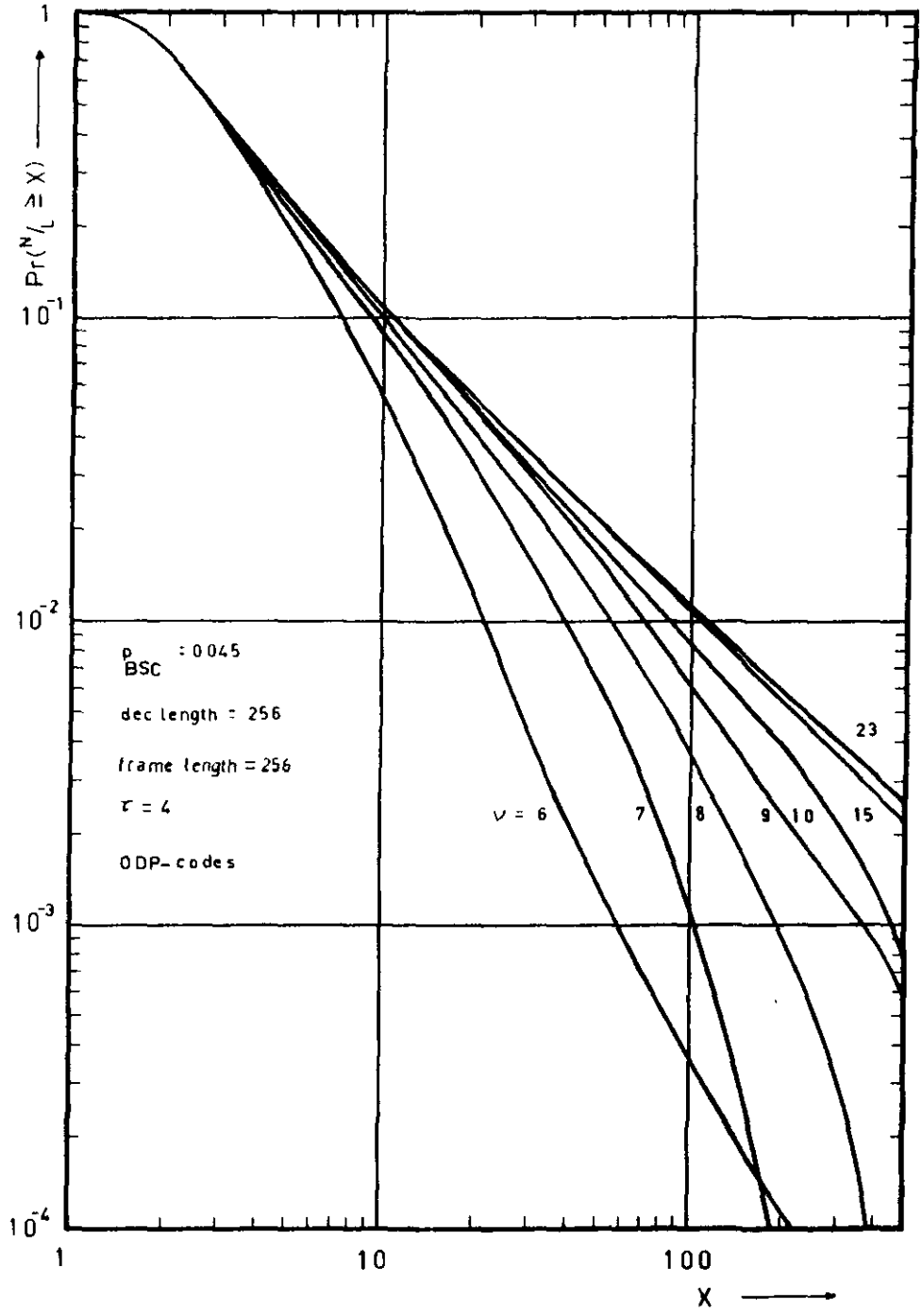


Fig. 11 Various ODP codes for decoding length 256.

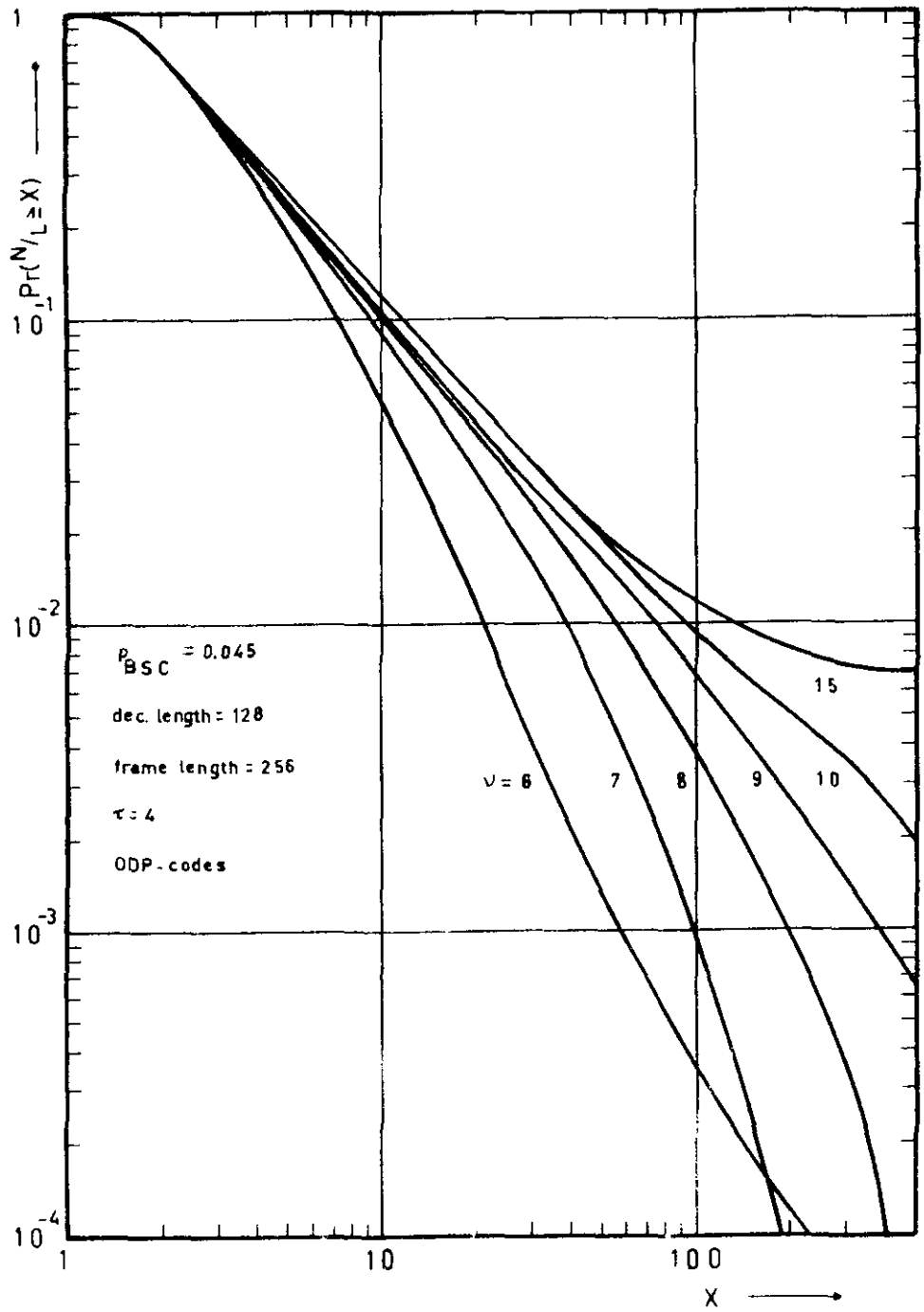


Fig. 12 Various ODP codes for decoding length 128.

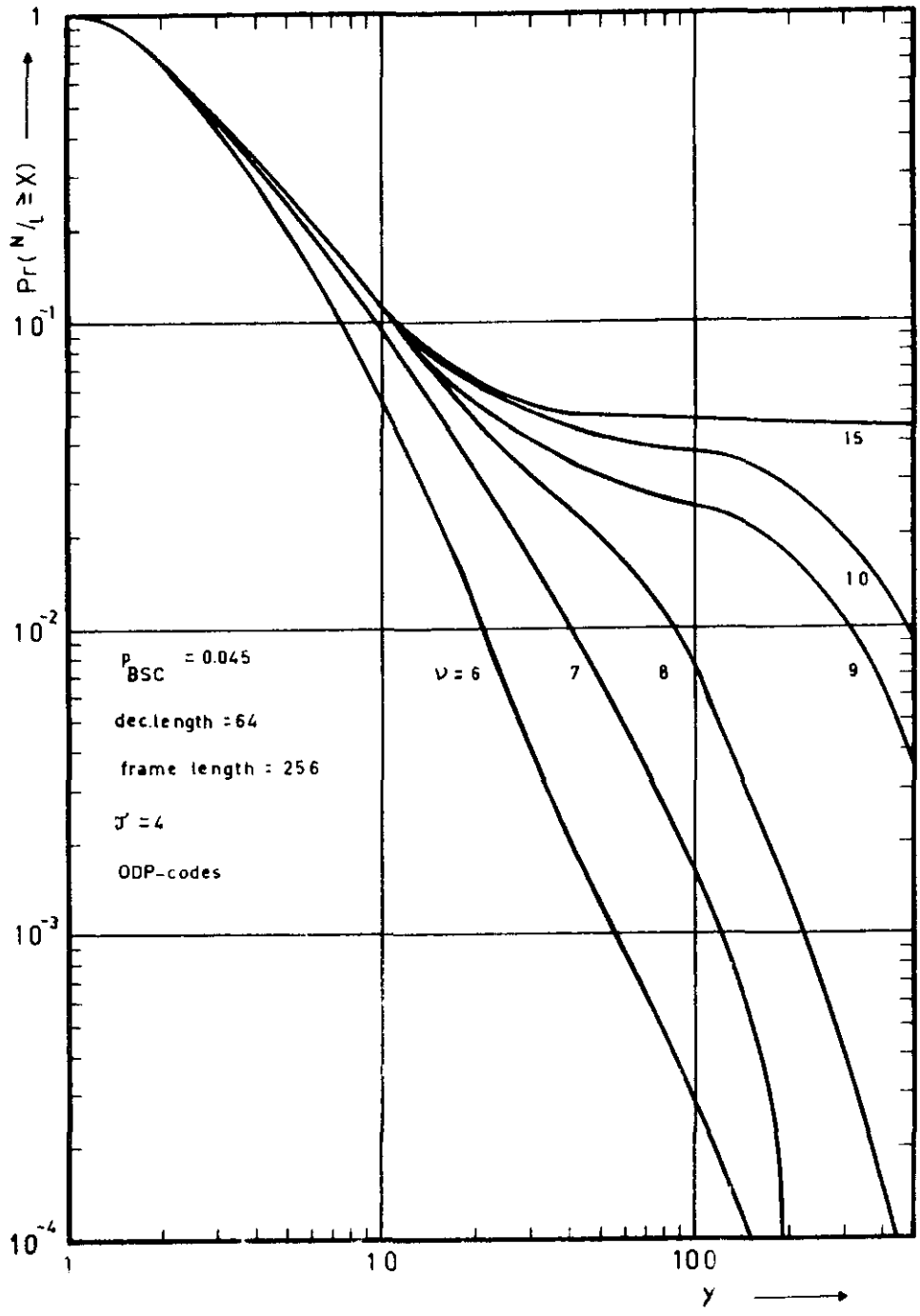


Fig. 13 Various ODP codes for decoding length 64.

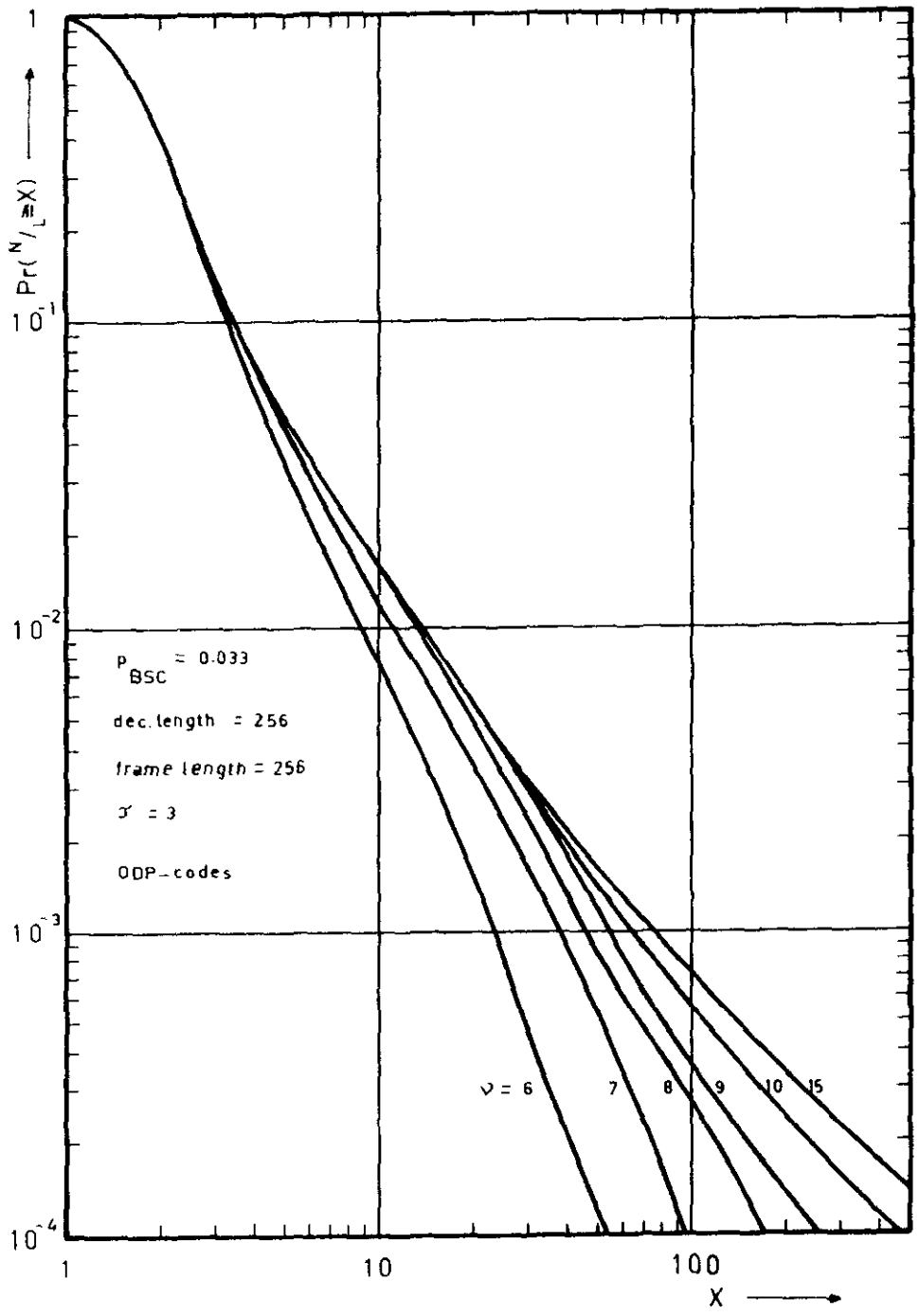


Fig. 14 Various ODP codes for $p_{BSC} = 0.033$.

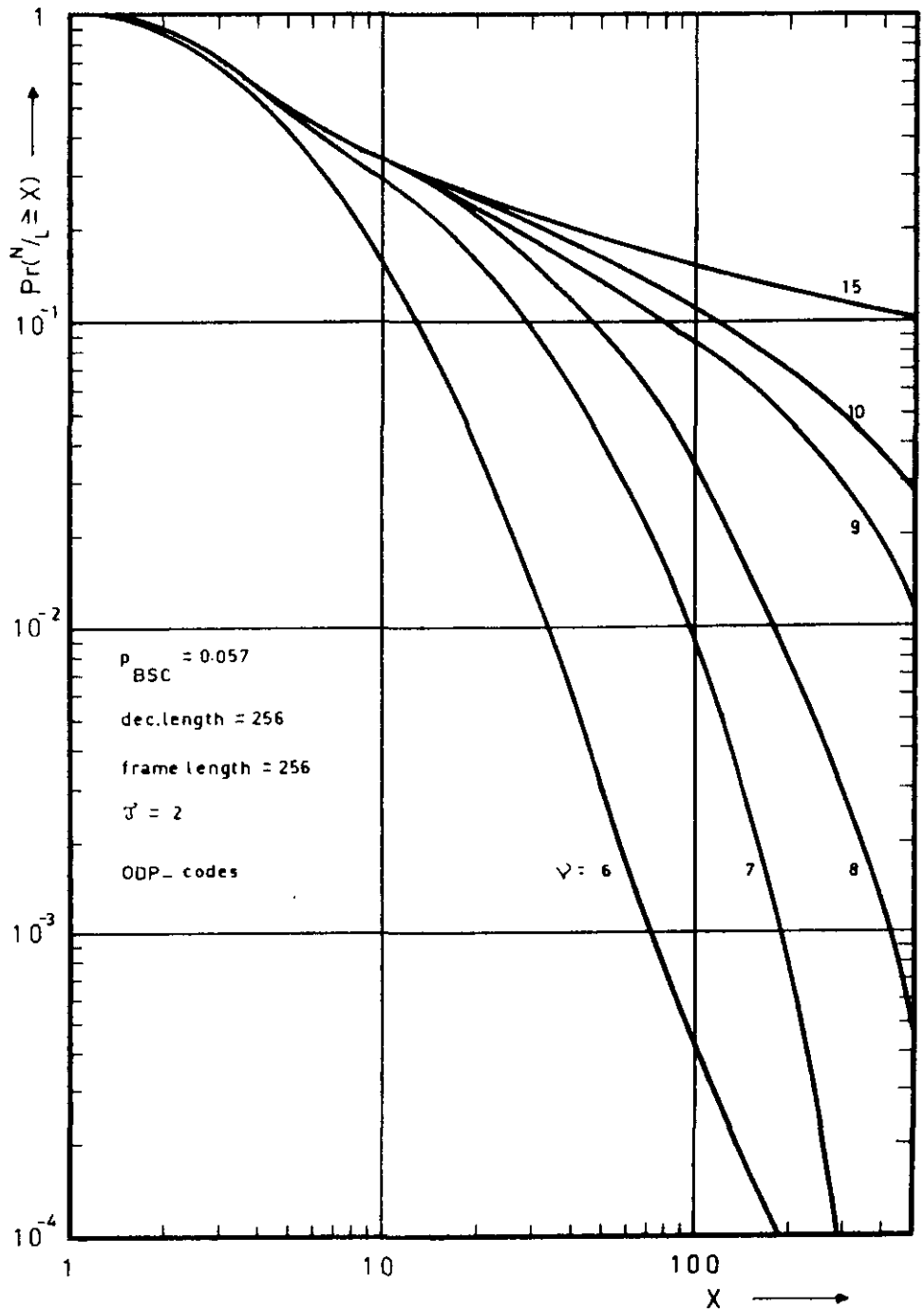


Fig. 15 Various ODP codes for $p_{BSC} = 0.057$.

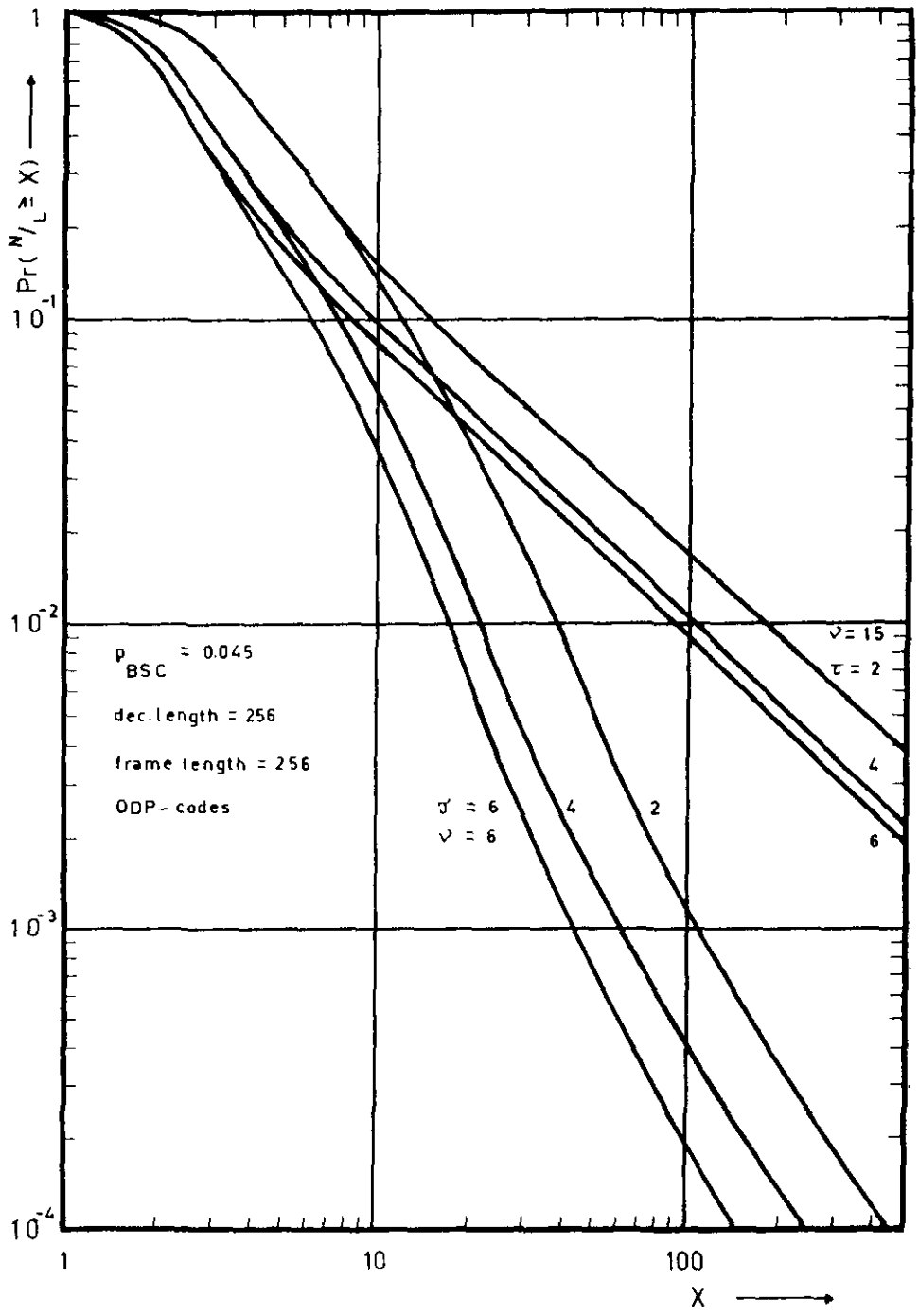


Fig. 16 Two ODP codes for some values of τ.

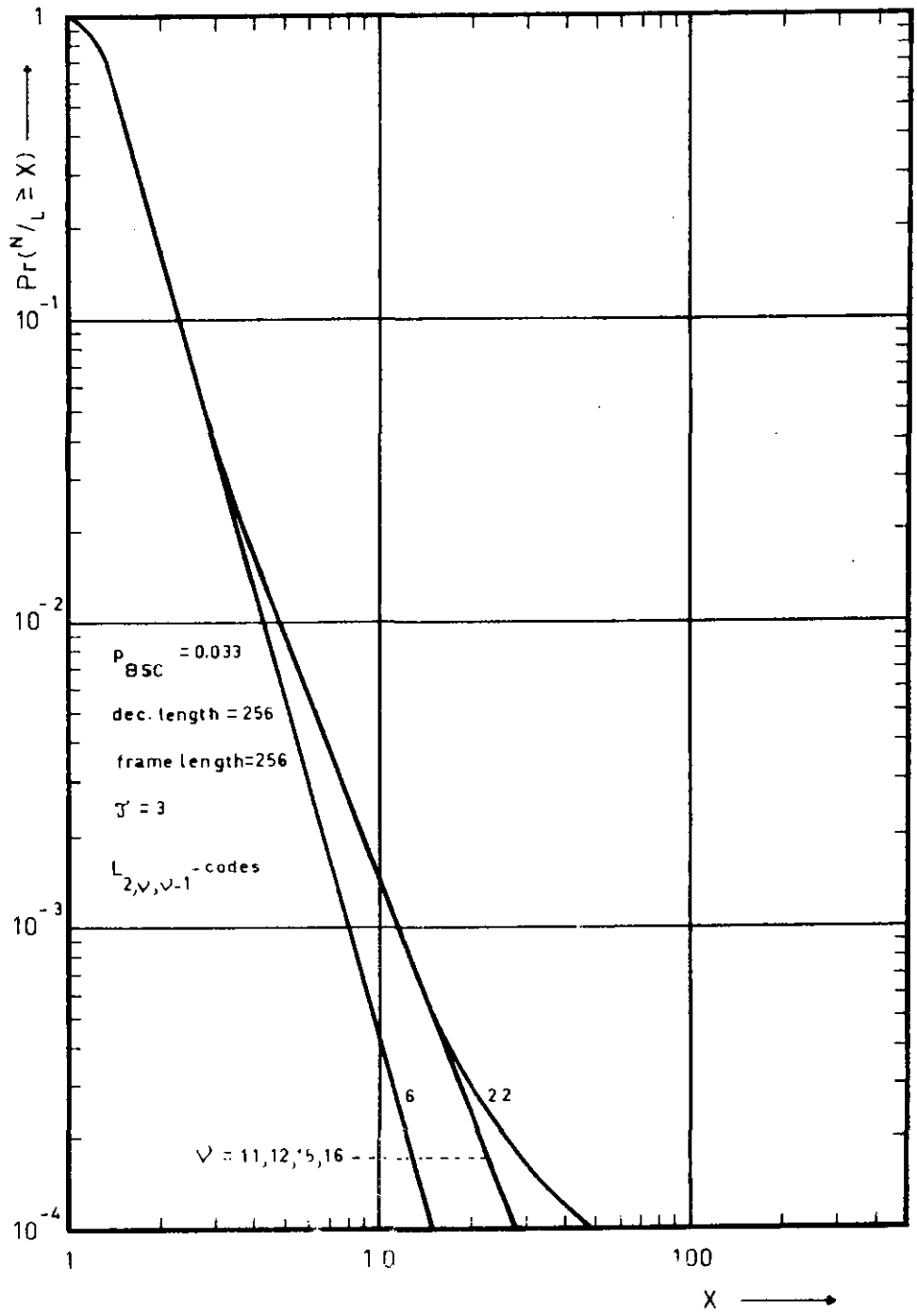


Fig. 17 Various $L_{2,\nu,\nu-1}$ codes for p_{BSC} = 0.033.

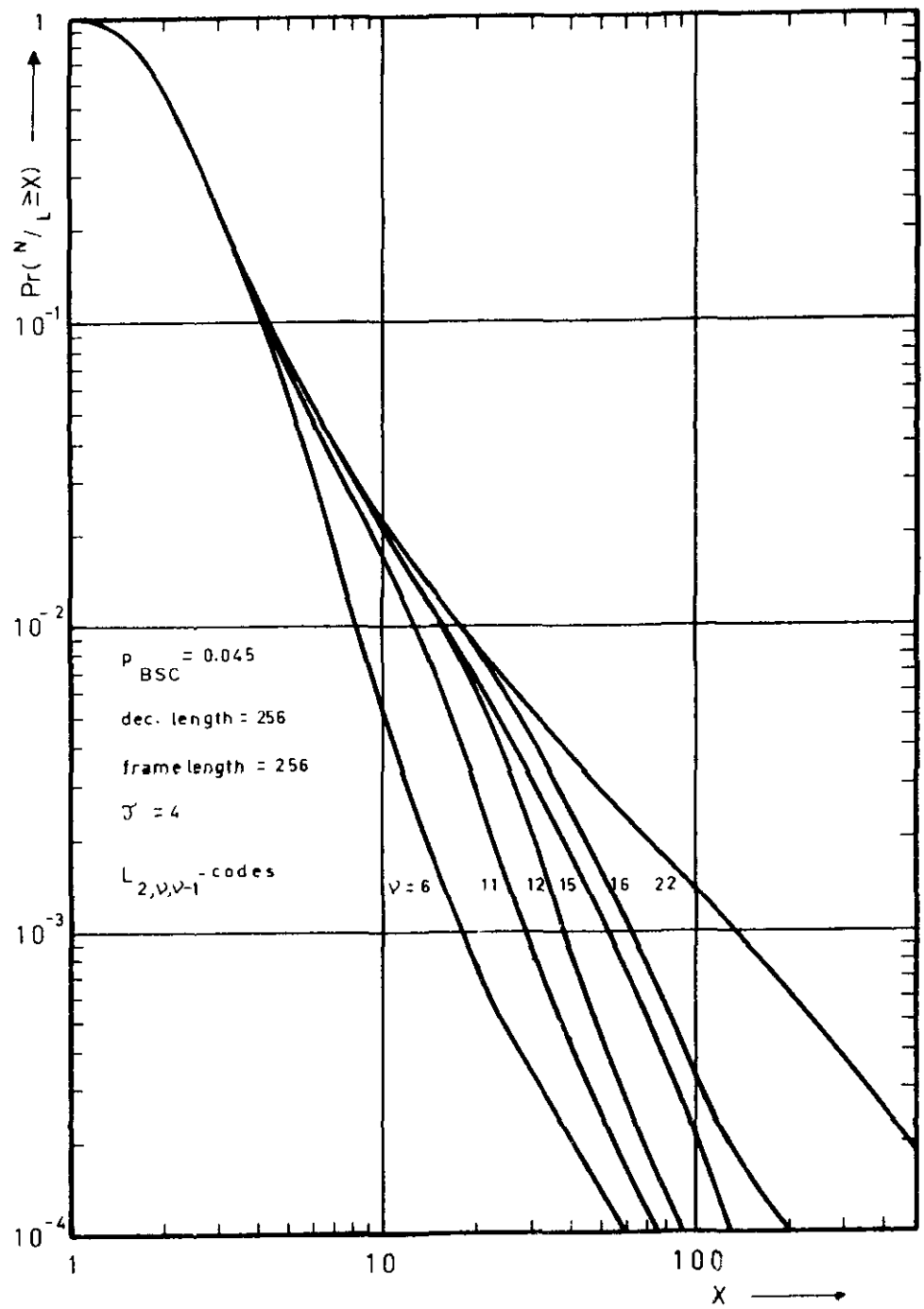


Fig. 18 Various $L_{2,\nu,\nu-1}$ codes for $p_{BSC} = 0.045$.

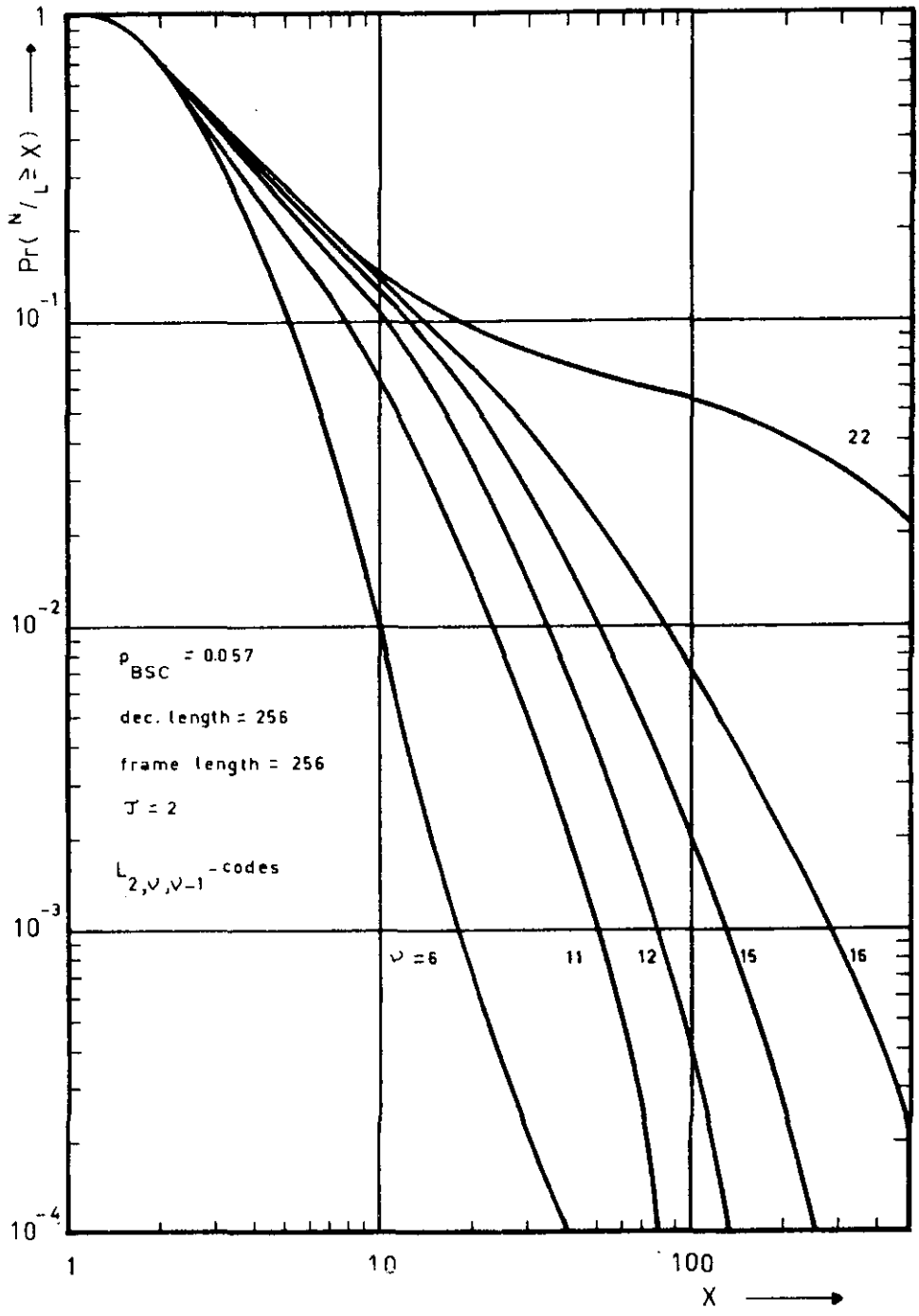


Fig. 19 Various $L_{2,\nu,\nu-1}$ codes for p_{BSC} = 0.057.

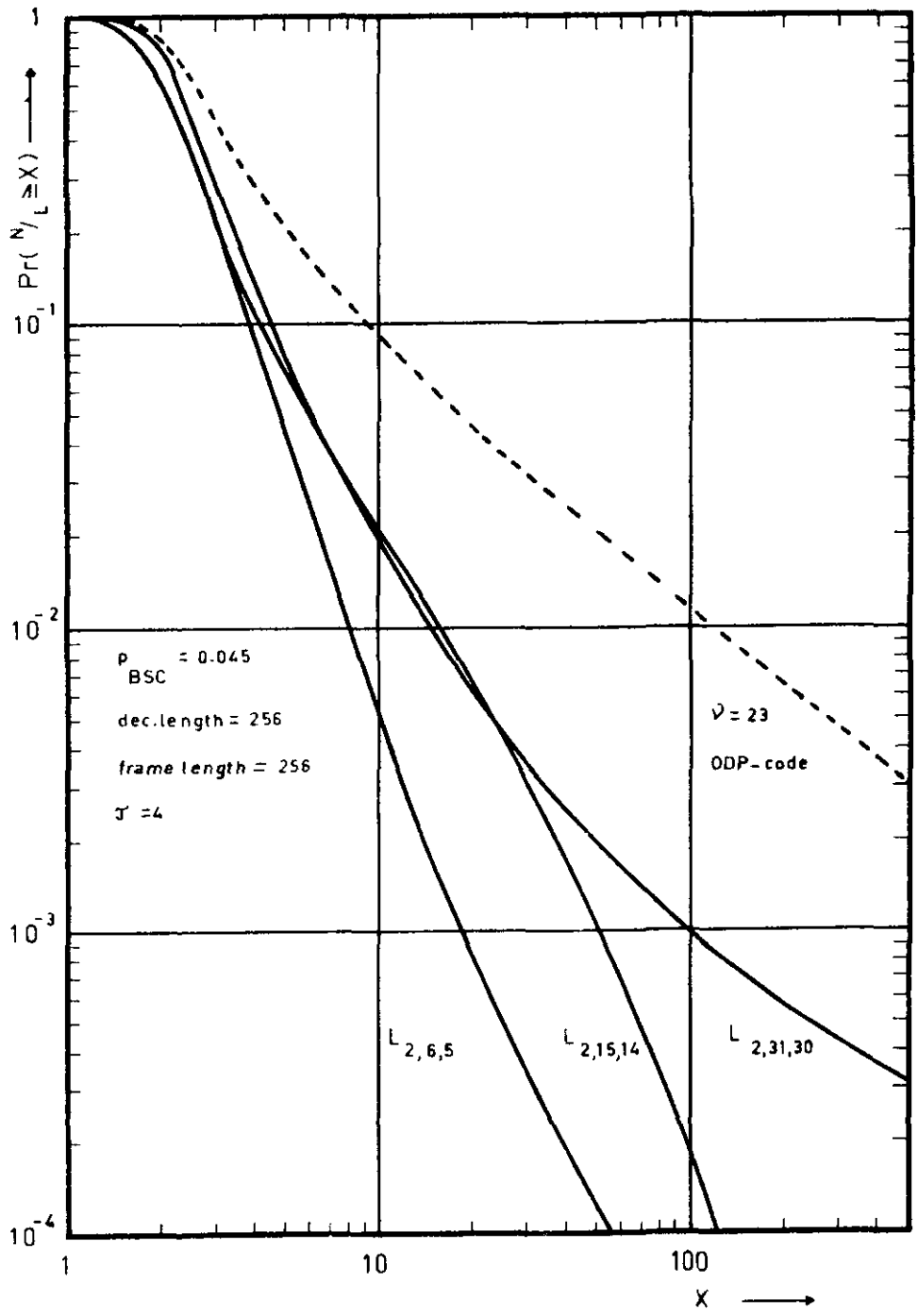


Fig. 20 Comparison between some $L_{2,\nu,\nu-1}$ codes and a ν= 23 ODP code.

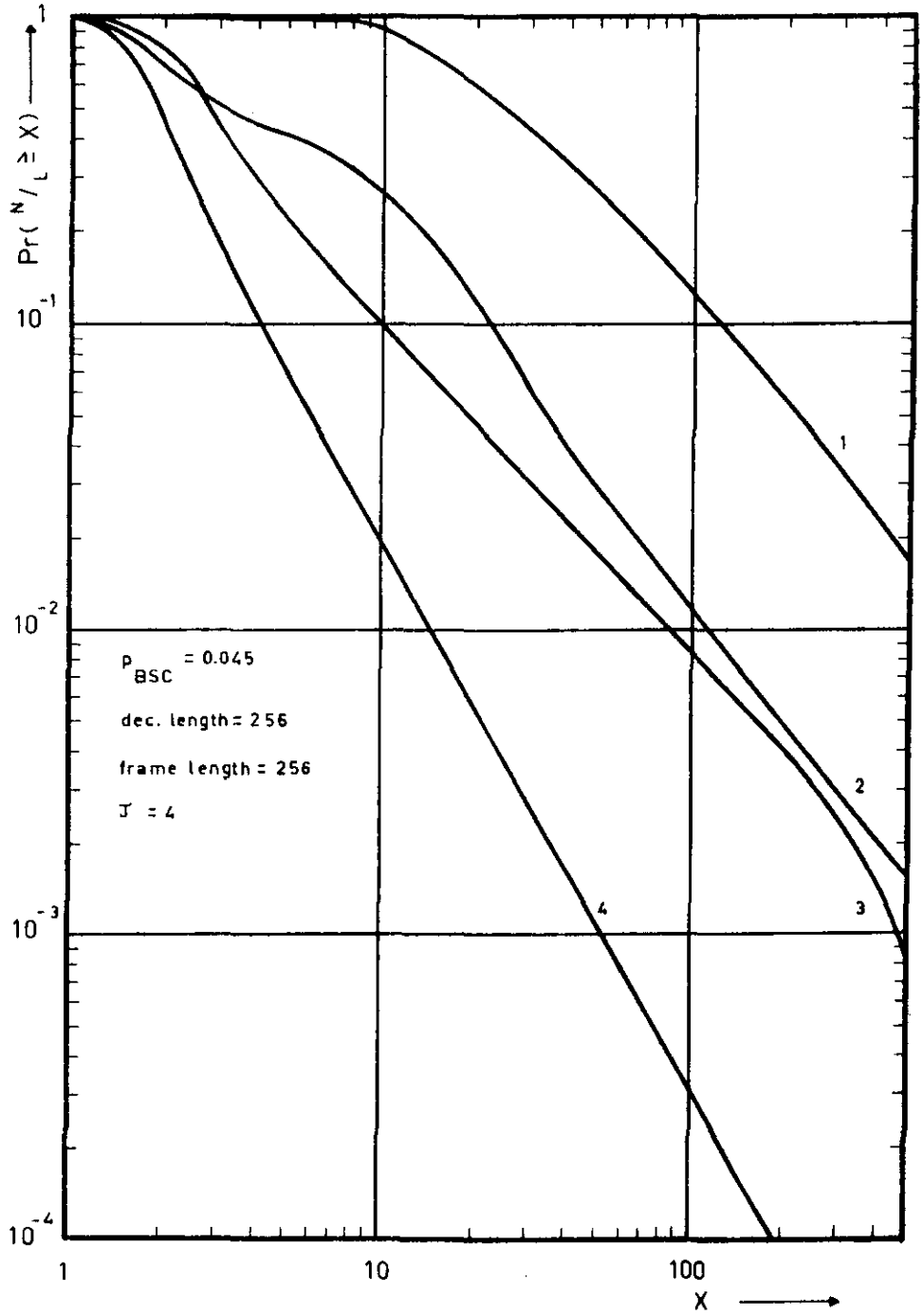


Fig. 21 $L_{2,15,14}$ code under different circumstances.

considered. Lines 3 and 4 are the optimal results for the ν=15 ODP code and the ν=15 code from $L_{2,15,14}$, respectively.

In Tables VII and VIII, we compare several $L_{2,\nu,\ell}$ codes with ODP codes. We define an erasure to be the event where N/L > 500. Furthermore, a frame is in error if it contains decoding errors. Observe a slight increased frame error probability for the codes in $L_{2,\nu,\nu-1}$. If we decode these codes according to "metric equivalence" classes, then this error probability decreases at the cost of an increased erasure probability. However, for long codes, with a large d_{free}, the measurements were error free. From Tables VII and VIII, one can conclude that there is a reduction in erasure probability with a factor of 10, and a factor of about 2.5 in the average number of computations when for instance the ν=23 ODP and the ν=31, $L_{2,31,30}$ code are compared.

Table VII

Comparison of several ODP and $L_{2,\nu,\nu-1}$ codes for p_{BSC} = 0.033

p_{BSC} = 0.033 dec.length = 256 τ = 3

ν	type	d_{free}	erased frames	frames in error	bit errors	N/L
6	ODP	10	0	564	3839	2.2
10	ODP	14	0	22	160	2.5
15	ODP	18	2	0	0	2.6
23	ODP	25	2	0	0	2.6
6	$L_{2,6,5}$	9	0	1051	5489	1.7
16	$L_{2,16,15}$	14	0	69	597	1.9
22	$L_{2,22,21}$	19	0	2	31	1.9
31	$L_{2,31,30}$	23	0	0	0	2.0

Table VIII

Comparison of several ODP and $L_{2,\nu,\nu-1}$ codes for $p_{BSC} = 0.045$

$p_{BSC} = 0.045$ dec. length = 256

ν	type	d_{free}	τ	erased frames	frames in error	bit-errors	N/L
6	ODP	10	4	0	2468	20817	4.1
10	ODP	14	2	58	121	1411	10.7
10			4	19	159	2023	7.1
10			6	17	167	2324	6.1
15	ODP	18	4	53	2	100	8.1
23	ODP	25	4	75	0	0	8.3
6	$L_{2,6,5}$	9	4	0	3732	24132	2.6
16	$L_{2,16,15}$	14	2	0	368	4749	4.8
16			4	0	411	5279	3.1
16			6	0	451	5829	2.6
22	$L_{2,22,21}$	19	4	4	15	323	3.4
31	$L_{2,31,30}$	23	4	8	0	0	3.5

In Table IX, we compare the number of additional computations in order to decode a frame with the indicated noise vector sequence starting from position 128. Both codes are ν=6 codes with an optimized distance profile, and a free distance of 9 for the $L_{2,6,5}$ code and a d_{free} of 10 for the ODP code. The code generators are g1=113, g2=153 and g1=135, g2=163, for the $L_{2,6,5}$ and the ODP code, respectively. In Table X, the instances are given where the decoder gives decoding errors. Accidentally, both codes give the same Table. In Table XI, we compare two ν=15 codes. Again, one is an ODP code, whereas the other is a code from $L_{2,15,14}$. Both codes were error free. For the ODP code g1=103745, and g2=164133. For the $L_{2,15,14}$ code, g1=104253 and g2=144253.

Table IX

Comparison between, two $\nu = 6$ codes in decoding a specific noise vector sequence, for an ODP, and an $L_{2,6,5}$ code.

n_2 \ n_1	000	001	010	011	100	101	110	111
000	0	8	4	32	8	26	22	80
001	8	384	32	662	26	520	80	1124
010	4	32	324	782	22	80	892	1200
011	32	662	782	2740	80	1140	1192	4438
100	8	26	22	80	384	498	1038	1580
101	26	520	80	1116	498	4678	1476	3148
110	22	80	892	1180	1038	1552	2074	3854
111	80	1116	1180	4438	1572	3148	3854	2698
000	0	8	4	28	8	22	20	76
001	8	32	28	84	22	66	76	120
010	4	28	42	84	20	86	64	118
011	28	84	84	174	78	120	118	428
100	8	22	20	74	32	98	116	284
101	22	66	74	120	98	248	258	298
110	20	74	64	118	116	264	222	370
111	74	120	118	428	290	298	370	274

Table X

Decoding errors for the patterns of Table IX

n_2 \ n_1	000	001	010	011	100	101	110	111
000	0	0	0	0	0	0	0	0
001	0	0	0	0	0	0	0	0
010	0	0	0	0	0	0	0	0
011	0	0	0	4	0	0	0	3
100	0	0	0	0	0	0	0	0
101	0	0	0	0	0	4	0	3
110	0	0	0	0	0	0	4	3
111	0	0	0	3	0	3	3	3

Table XI

Comparison between $\nu = 15$ codes in decoding a specific noise sequence for ODP and $L_{2,15,14}$ code.

n_2 \ n_1	000	001	010	011	100	101	110	111
000	0	8	4	44	8	24	24	222
001	8	122	46	182	24	176	160	468
010	4	44	92	776	34	108	244	906
011	46	196	966	1656	134	972	1014	6012
100	8	24	24	220	122	472	1210	9444
101	24	178	158	522	266	2532	4576	29482
110	34	102	260	1002	1316	2678	2388	24548
111	112	1038	1096	25418	2522	29729	15090	29729
000	0	8	4	28	8	22	20	74
001	8	42	28	50	22	42	74	66
010	4	28	32	82	20	74	68	132
011	28	50	82	300	74	66	132	1032
100	8	22	20	74	42	42	58	132
101	22	42	74	66	42	202	132	1130
110	20	74	68	132	58	132	364	1168
111	74	66	132	1032	132	1130	1168	1294

IV. Restricted Viterbi Decoding

The application of the Viterbi decoding algorithm [2] for convolutional codes is limited to short constraint length codes. The reason for this is the exponentional growth in the number of different path- and metric memory registers connected with encoder states. In this Chapter we give a decoding scheme with reduced memory necessity, and hence, in this respect, decoder complexity. We are able to use the memory in such a way that also the computing complexity can be kept low.

Every decoding step, the Viterbi decoder for a memory length ν encoder extends $2^{**}\nu$ paths, or information sequence estimates. These estimates differ at least in the last ν stages. The metrics of the extended paths depend on the Hamming distance between a received vector sequence and an estimated code vector sequence. For rate 1/2 codes, there are $2^{**}(\nu+1)$ successors, from which the best, i.e. lowest metric, of each two estimates ending in the same encoder state are retained. For short constraint lengths $\nu<8$, the implementation is feasible. For longer constraint lengths the obtainable free distance increases, and thus, the undetected error rate can be decreased. Hence, one would like to implement decoders for $\nu>7$. One solution is found in sequential decoding algorithms like Fano or Stack decoding. However, the disadvantages of these decoding algorithms are the variable amount of the number of computations, and the need for a feedback link. Another solution can be found if we sacrifice the optimality of the Viterbi decoding algorithm.

Suppose, we make the following restrictions on the Viterbi decoding algorithm. 1). Extend a maximum of $m<2^{**}\nu$ paths. After each extension procedure, retain the m best, i.e. with lowest metric. This restriction directly influences the decoding complexity, and a few remarks must be made. As the information sequence may take on every value with equal probability, all encoder states have the same probability of occurance. This forces the last ν digits of the extended sequences not to be fixed. For, if they were, for values of $m<2^{**}\nu$, certain encoder states can never be reached, resulting in a useless decoder. Now, we need a sorting algorithm that selects the best m paths from a list of $2*m$ candidates. This sorting algorithm strongly influences the decoding complexity, and might be of a complexity proportional to $m^{**}2$. Hence, only practical for very small values of m. However, the metric values of the paths are known to be positive, and take on values close to zero. We might therefore be able to profit from this knowledge, as will be shown later. 2). The second restriction is that only those paths are extended that differ from the minimum metric path in the last ν-positions. This restriction could lead to paths ending in the same state to be present in the decoder. Upon extension, these paths again give rise to paths ending in the same state. This effects the maximum number of different paths that can be extended, and hence, could increase the undetected error rate. However, by leaving out the requirement that all paths must be different in the last ν stages, we greatly influence the computational complexity of the decoder. 3). The last restriction is that we only extend paths for which the metric is smaller than some fixed value, say 20 or 25. Again this may increase the undetected error rate.

For, a path with a temporarily high metric could be the path to be decoded later on. As we are dealing with codes with a free distance of about 20, the influence is kept small.

We will see that there is an implementation for the restricted Viterbi decoder for which the complexity is proportional to m. One of the interesting questions arises how the undetected error rate behaves as a function of m. We now give a description of the restricted algorithm, for a software as well as for a hardware realization. In the decoding algorithm we use two memories. One of them is the transmitting memory, whereas the other plays the role of a receiving meomory. The estimates present at one memory on a certain time instant are extended and transferred to the receiving memory. The locations where the successors are stored are calculated in such a way that the computational complexity can be kept proportional to m. The following time instant, the process goes in the opposite direction.

Suppose we have the availability of two addressable memories M^o and M^e, respectively, with a certain word length. This word length can be taken equal to the decoding length. Each memory can be divided into buckets $B_0^o, B_1^o, B_2^o, \ldots$, and $B_0^e, B_1^e, B_2^e, \ldots$, of variable size. On odd time instants bucket B_j^o contains the estimates with a metric value equal to j. The same can be said for bucket B_j^e on even time instants. The bucket which contains the estimate with a metric value equal to the minimum value is called B_{min}, whereas the estimates with maximum metric are stored in B_{max}. On odd time instants,

the successors of an estimate from B_j^o are stored in memory M^e in a bucket corresponding with the metric value of the successor. For even time instants, the operations go the other way around. For binary 1/2 codes, there are two possible successors for each estimate, see Figs. 1 and 2. The metric value of the successors is equal to the old metric plus the Hamming distance between a received pair of digits and the code pair estimate corresponding with the respective transitions. It is easily verified that for a code with generators g_1 and g_2 that have a constant term, these distances are equal to 0 and 2, or both equal to 1. Hence, an estimate from M^o in bucket B_j^o can give rise to a successor in buckets B_j^e and B_{j+2}^e, or two successors in bucket B_{j+1}^e. If we subtract the minimum metric value, min, of the metrics of the estimates of M^o, then an estimate of B_j^o can give rise to a successor to be stored in B_{j-min}^e, $B_{j+2-min}^e$, or $B_{j+1-min}^e$. Note that the minimum metric is found easily by searching for the first non empty bucket. Based on the above considerations, we can calculate the starting addresses P_j^o and P_j^e of the buckets of the receiving memory. For, the pointer P_j^e of bucket B_j^e must be greater or equal to pointer P_{j-1}^e plus the maximum number of successors that can be stored in bucket B_{j-1}^e. Hence,

$$P_j^e = P_{j-1}^e + |B_{j-1+min}^o| + 2\,|B_{j-2+min}^o| + |B_{j-3+min}^o| \,, \qquad (7)$$

for the odd time instants. The indexes e and o interchange for the even time instants. In (7) $|B_j^o|$ denotes the initial number of estimates stored in bucket B_j^o. Note that if m paths are stored in M^o, the maximum size of memory M^e that can be declared using the above

pointer technique is equal to 4*m. In Fig. 22 we give a flowchart of the decoding algorithm using the memory organization from above.

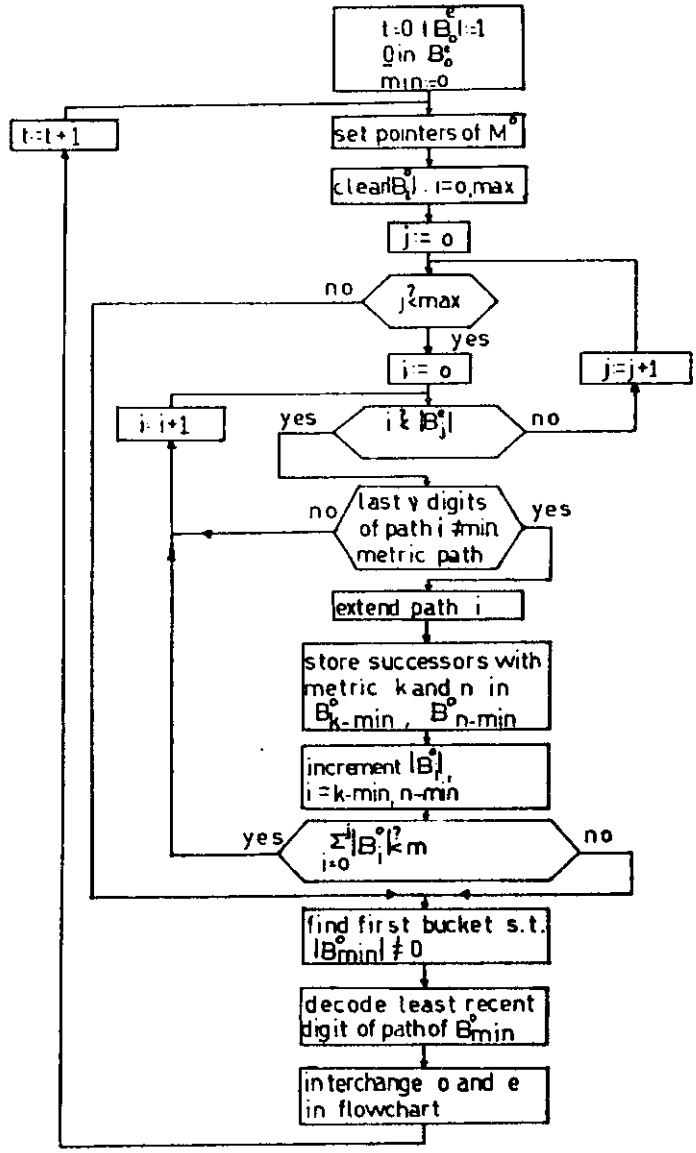
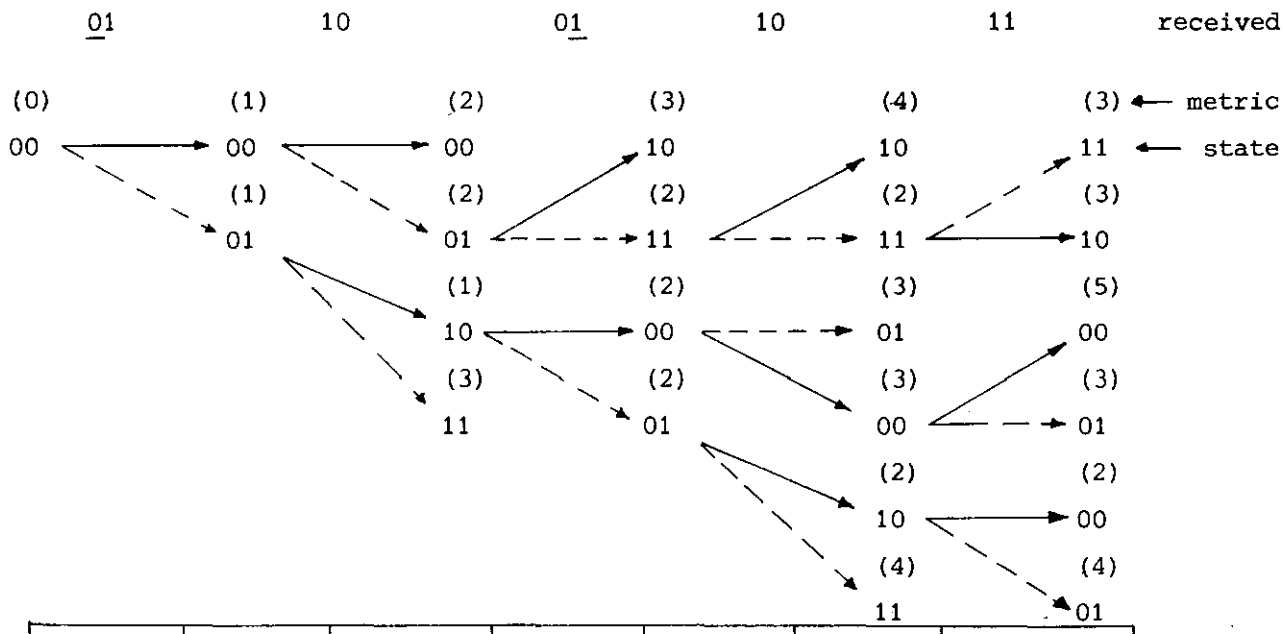


Fig. 22 Flowchart of the restricted decoding algorithm.

In Fig. 23 we give the value of the relevant pointers and the number of paths stored in each bucket, for the example of Fig.3. We have taken m=3. If there are more than 3 estimates available in the memory, we correct the number of paths available in the buckets to equal m.



P_j^e	$\|B_j^e\|$	P_j^o	$\|B_j^o\|$	P_j^e	$\|B_j^e\|$	P_j^o	$\|B_j^o\|$	P_j^e	$\|B_j^e\|$	P_j^o	$\|B_j^o\|$	bucket
0	1	0	0	0	1	0	0	0	2	0	1	0
		1	2	2	2	1	3	3	2→1	2	3→2	1
		3	0	6	1→0	5	1→0	9	0	7		2
		4	0	8	0	10	0	12		11		3
						12				12		4

Fig. 23 Example of the operations of the decoder.

We have also implemented the restricted decoding algorithm using the structure of the class $L_{2,\nu,\ell}$. As in the restricted decoder all paths have equal length, we have to recalculate (5). We therefore define $M[\underline{c}_{[0,j+\ell)}]$ to equal the Hamming distance between a code sequence $\underline{c}_{[0,j+\ell)}$ and a received sequence $\underline{r}_{[o,j+\ell)}$. The metric $M[\underline{c}'_{[o,j+\ell)}]$ corresponding with the estimate $i'_{[o,j+\ell)} = i_{[0,j+\ell)} \oplus i^{\Delta}_{[o,j+\ell)}$ follows easily from the metric $M[\underline{c}_{[o,j+\ell)}]$. To wit

$$M[\underline{c}'_{[o,j+\ell)}] = M[\underline{c}_{[o,j+\ell)}] + d_H[(11) : \underline{n}_{[j,j+1)}] \qquad (8)$$

where

$$d_H[(11) : \underline{n}_{[j,j+1)}] = \begin{cases} +2 \text{ if } \underline{n}_{[j.j+1)} = (00) \\ -2 \text{ if } \underline{n}_{[j,j+1)} = (11) \\ 0 \quad \text{otherwise} \end{cases}$$

The extension of an estimate to two successors is done in the same way as described in Chapter I. From the representative of a class of 2**ℓ paths, two representatives for two classes of 2**(ℓ-1) are derived. The metric of the altered representative must be calculated according to (8). Again, as in Chapter I, we extend each representative with a noise estimate that is an element from {(00),(01)}. If the noise estimate corresponding with a transition to a successor of a respective representative equals (00), then the noise estimate following from the extension of the other representative equals (01). The reason for this is the complementarity of the (ℓ+1)th connections of the generator polynomials. As the component $i^{\Delta}_{[j,j+1)}$

is unequal to zero, the transitions resulting from both representatives must differ by a component equal to (01). Using (8), we can calculate the overall metric increments for a particular successor. These increments are equal to 0 and 3, 0 and 1, or 1 and 2, respectively. Using these values, the pointers for the receiving memory are set according to

$$P_j^e = P_{j-1}^e + |B_{j-1+min}^o| + |B_{j-2+min}^o| + |B_{j-3+min}^o| + |B_{j-4+min}^o| \quad (9)$$

Again, the size of the declared memory is equal to 4*m.

Before presenting simulation results for the decoding algorithm, we compare a ν=15 ODP code with a code from $L_{2,15,14}$. In Table IX we give the Hamming weight distribution of paths diverging from the

Table IX. Weight distribution over the unmerged code word span, ν = 15, ODP code.

length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	total
distance															
2	1														1
3		2	1												3
4			2	3	2										7
5			1	3	3	6	3								16
6				1	6	6	9	11	6	1					40
7				1	4	8	13	18	18	21	12	5			100
8						8	15	18	35	36	42	36	29	12	231
9					1	2	13	30	42	58	73	84	83	80	466
10						2	7	28	50	90	116	147	164	171	775

all zero path in the code tree of an ODP code as a function of the length. In Table X we give the same distribution for a code in $L_{2,15,14}$ with an optimized distance profile. For the last code, we used the tree structure of $L_{2,\nu,\ell}$. Note that not only the distance profiles are quite different, but also the total number of paths or representatives at a certain Hamming distance over the unmerged span with the all zero sequence. This latter property makes the class $L_{2,\nu,\ell}$, attractable to implement in the restricted decoding algorithm, as this algorithm only keeps a small number of paths in consideration.

Table X. Weight distribution over the unmerged code word span, $\nu = 15$, $L_{2,15,14}$

length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	total
distance															
2															0
3	1														1
4		1													1
5		1	2	1											4
6			1	1	1										3
7				2	4	3	1								10
8				1	3	3	5	4	3	2					21
9					1	4	8	8	4	5	2				32
10					3	5	9	10	11	9	8	6	1		62

In Figs. 24 and 25 we compare several codes from $L_{2,\nu,\nu-1}$, with some ODP codes. We have plotted the bit error probability as a function of m, for both the BSC transition probabilities 0.045 and 0.033. In Fig. 26 we give the performance of a code from $L_{2,16,15}$ for several values of m.

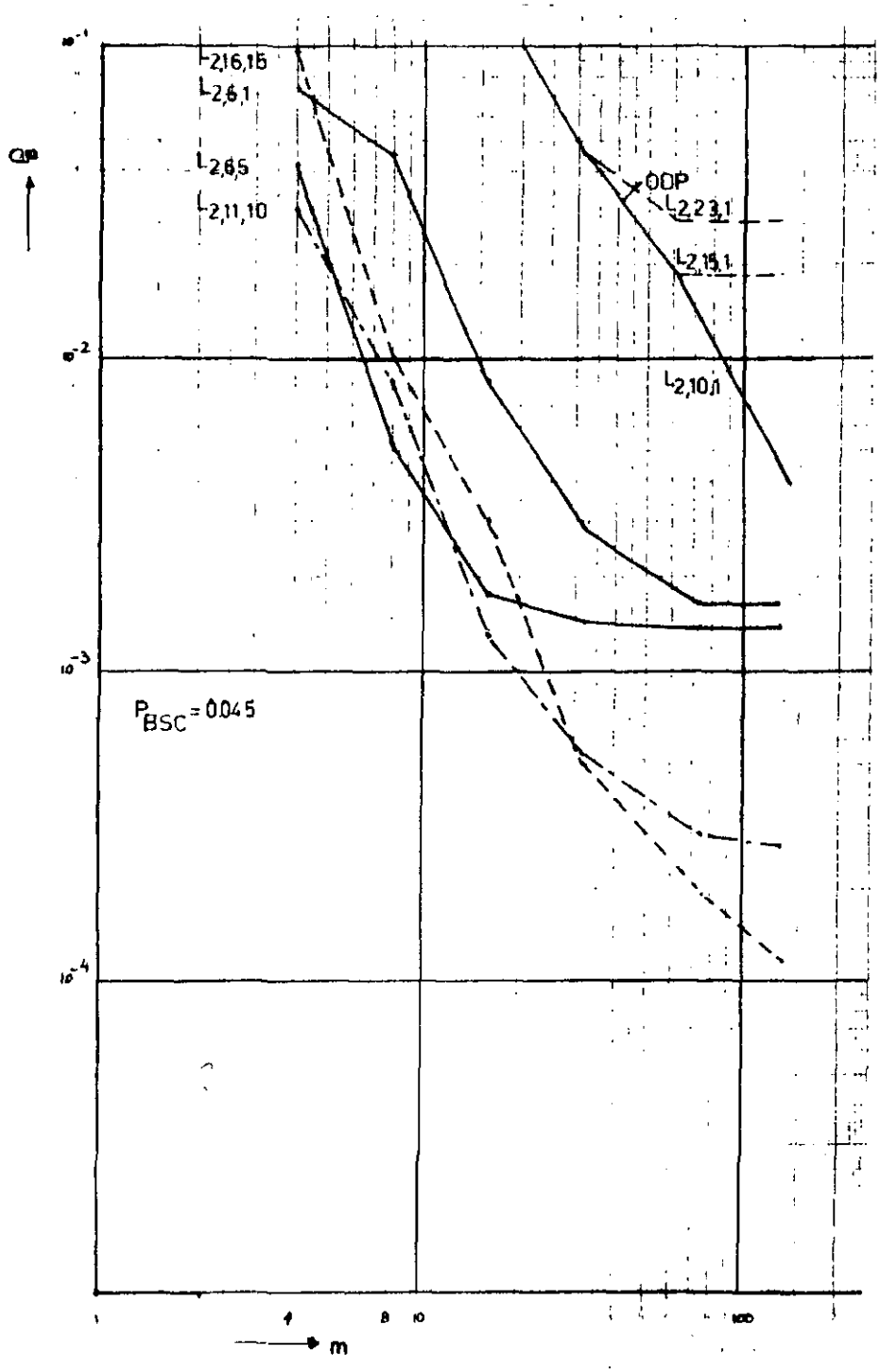


Fig. 24. Bit error probability as a function of m, for p_{BSC} = 0.045.

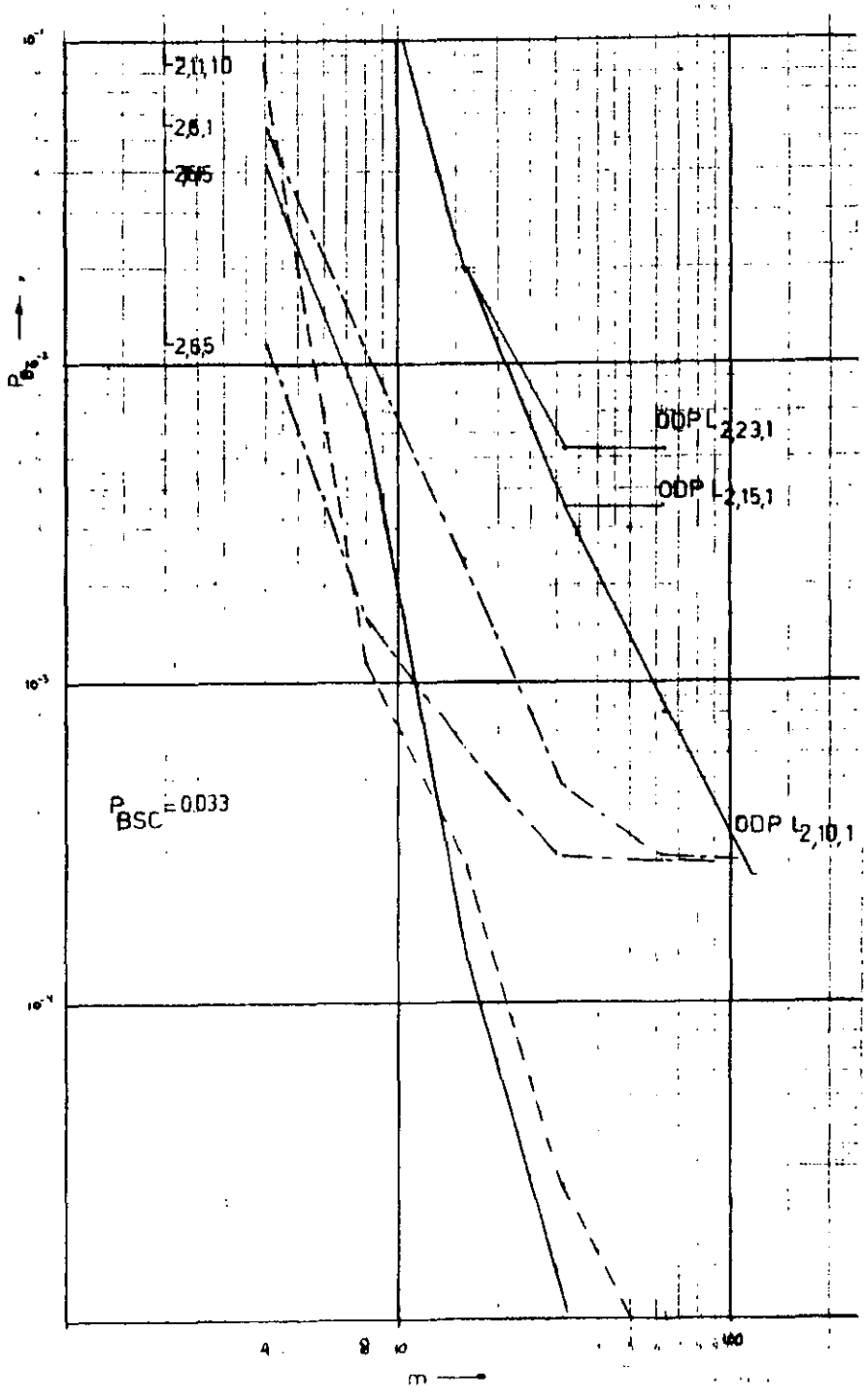


Fig. 25. Bit error probability as a function of m, for $p_{BSC} = 0.033$.

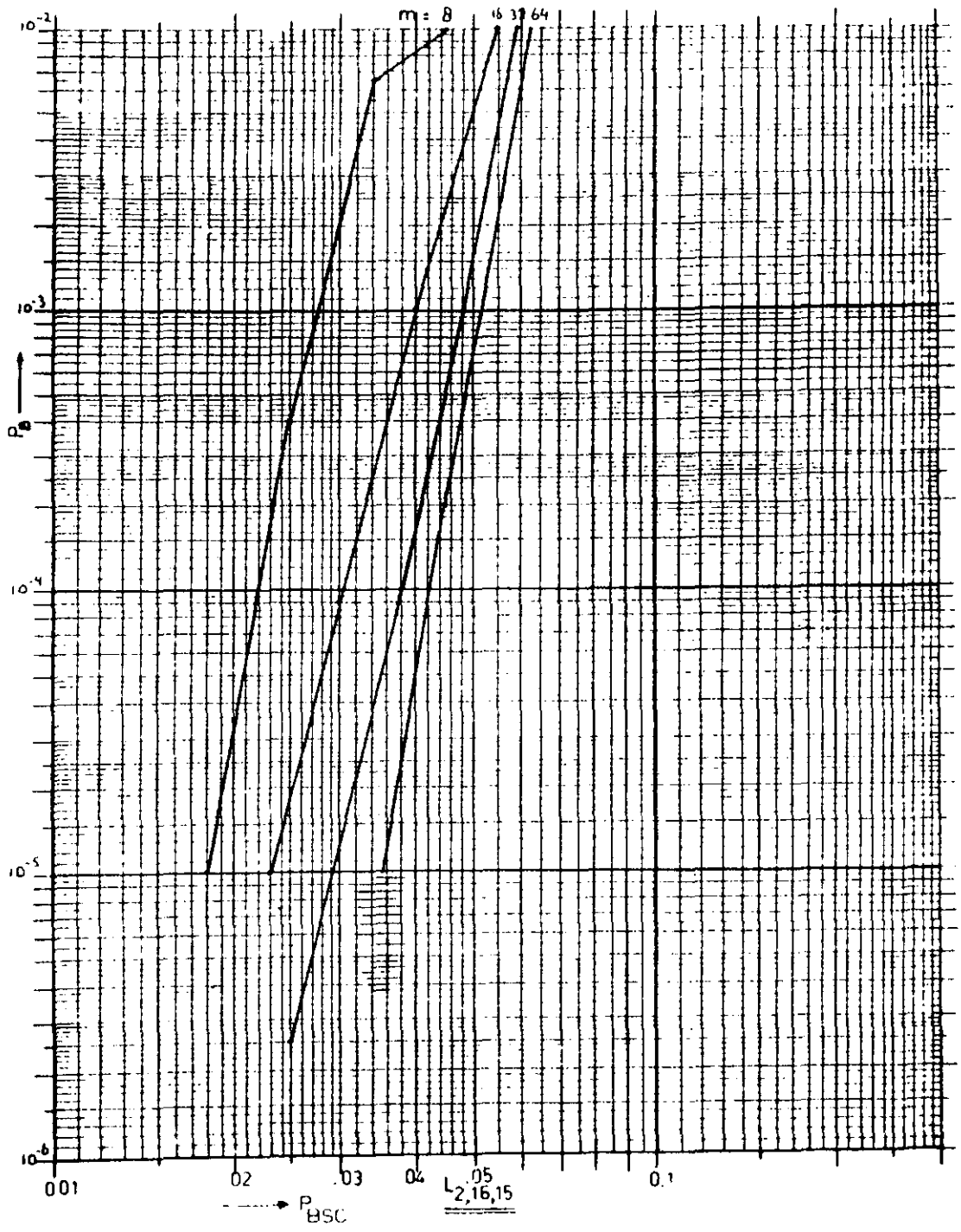


Fig. 26. Bit error probability as a function of p_{BSC}.

In the last part of this chapter we partly describe an implementation of a restricted Viterbi decoder employing a code from $L_{2,7,6}$. The value of m is taken equal to 8. The core parts in the hardware decoder are the sorting- and extension modules, which we first shortly describe.

As in the software implementation, we want to avoid complicated *storing and sorting routines. Therefore,* we store the paths, generated by the extension module (EXTD), on an address determined by the metric value of this path and the number # of paths already stored with the same metric value. More specifically:

write address = 8 x M + # + 1,

where we suppose that the memory has been split up into M domains of size 8. As we want to read out or write in different paths every time instant the decoder needs to, we have to update the number after each read or write cycle. Using this strategy, sorting speed depends on the time we need to find a specific metric domain and the appropiate value of # .

From simulation results, it followed that M=10 was sufficient for the specific code to be implemented. So, if we define M_c to equal the current value of the metric, and # M_c the current number of stored paths with metric M_c, then the read and write addresses are calculated as

$$\begin{aligned} \text{write address} &= 8 \times M_c + \#\, M_c + 1 \\ \text{read address} &= 8 \times M_c + \#\, M_c . \end{aligned} \qquad (9)$$

For every value of M_c, we have a memory that contains $\#\, M_c$. In practice, every write instruction of a path with metric M_c, $\#\, M_c$ increases by one. For each reading operation $\#\, M_c$ is decreased. For speed reasons, the required search for a path of lowest current metric should be done asynchronously. The "address available" moment, however, is controlled by the decoder. We now give an explanation for the address calculating module as given in Fig. 27.

i). The write cycle. The extension module generates a path with corresponding metric. The write address is generated as follows.

1. The metric value is switched on the BCD/DEC decoder which enables the corresponding counter / 3-state-latch doublet.
2. $U/\overline{D}$ high; EC pulsed:

   $\#M_c := \#M_c + 1$
3. EI pulsed:

   $\#M_c$ is read into the latch and available at its output.

   Meanwhile the metric value has been multiplied by 8 (shifted 3 positions to the left, hard wired) and passed to the adder.
4. The address is read from the output.

ii). The read cycle. The decoding computer asks for the address of a path having the smallest metric. This is easily found. Suppose at the current time, no paths with metrical value of zero are stored: $\#M_0 = 0$.

The carry output of counter 0 = 0. Suppose we have stored a number of paths with metric 1 so $\#M_1 \neq 0$. Therefore, the carry output of counter 1 = 1. The carry outputs of all other counters are don't cares to the priority encoder that thus generates a binary one at its output. This value is the wanted smallest metric. The value is also fed to the BCD/DEC decoder which enables the corresponding counter/3-state-latch doublet.

In time order:

0. $U/\overline{D}$ is low.
1. The switch is set in top position.
2. EI is pulsed:

   The latch is set to the $\#M_1$ value which is available on the bus line.
3. EC is pulsed: the counter is decremented by one.
4. If this leads to $\#M_1 = 0$; the priority encoder output changes to the smallest non zero counter.

2a. Meanwhile the metric value has been fed to the multiplier and adder.

3a. The read address and the metric of a path of smallest metric value are available.

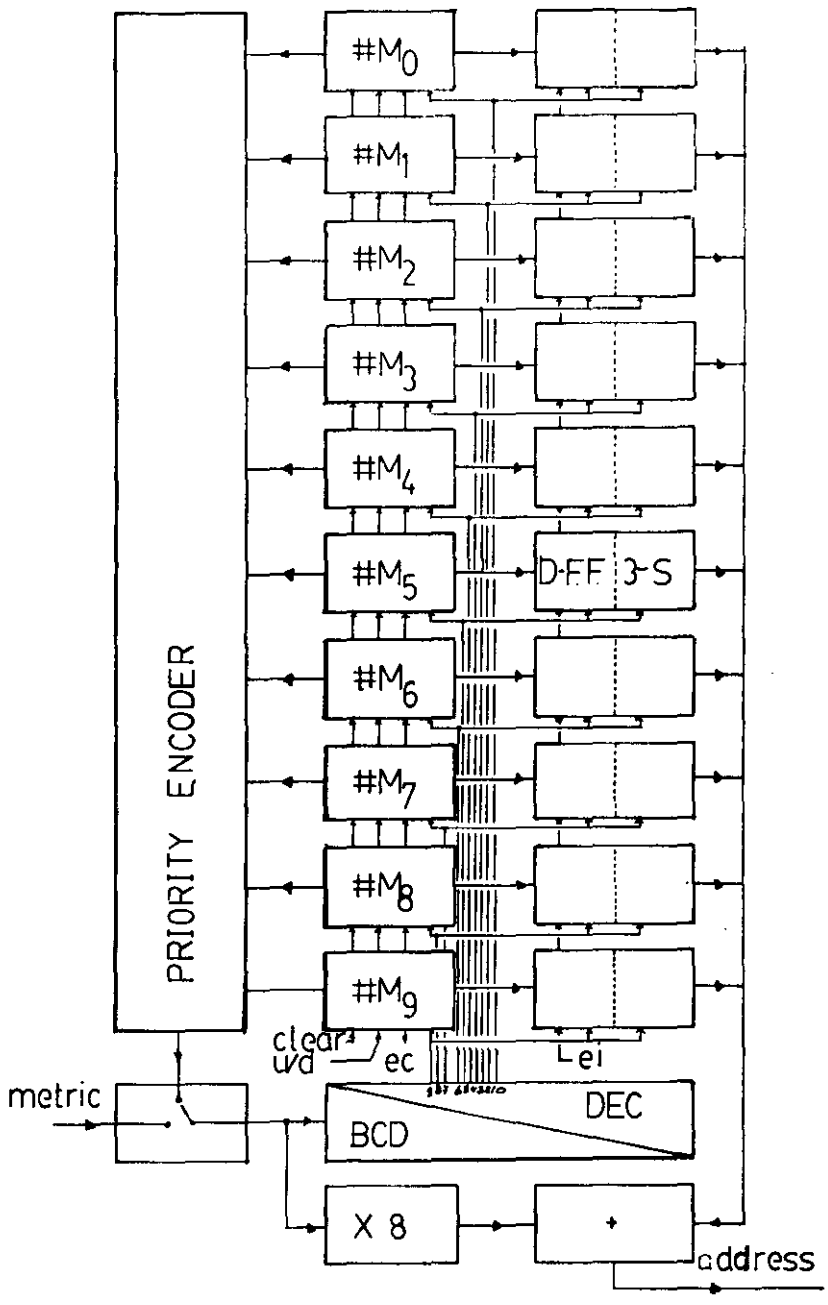


Fig. 27. Address computation.

In the EXTD module, we calculate the successors to a representative and the corresponding metrics, see Fig. 28.

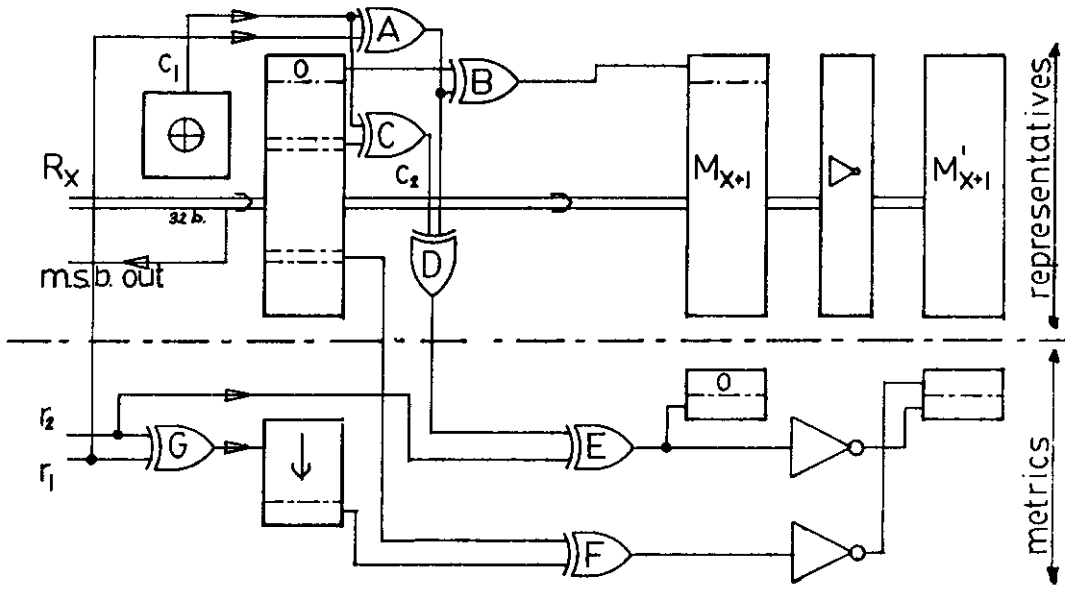


Fig. 28. Extension module (EXTD)

The extensions are done according to noise estimates from {(00),(01)}. If an extension with a digit equal to zero is invalid according to this criterion, then the zero is inverted to be a one. The determination of $\underline{n}_{[j,j+1)}$ according to (8) can be done easily by using a delayed version of $r_1 \oplus r_2$, and the properties of $L_{2,7,6}$, see Fig. 28. The sorting- and extension modules are the core parts in a hardware decoder that is to be described. The use of a code from $L_{2,7,6}$ is sufficient to demonstrate decoding feasibility. However, without changing the structure of the decoder, a

more powerful code could be used. To decrease decoding time, we employ "pipe-lining" of the respective data-streams. From Fig. 29 the circuit and the dataflow will be explained.

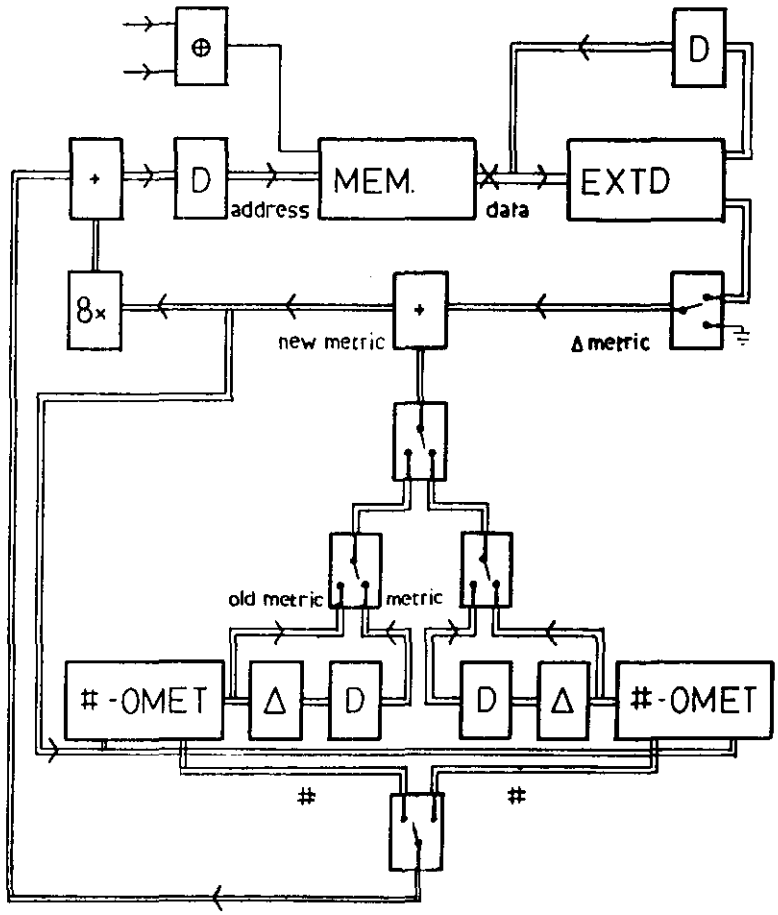


Fig. 29. Block diagram of the decoder.

One sub-cycle from a collection of eight consists of:

1. Locate the stored representative of

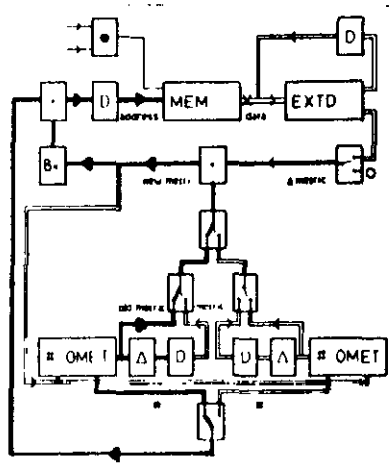
   current lowest metric:

   Proceeding according to the above the

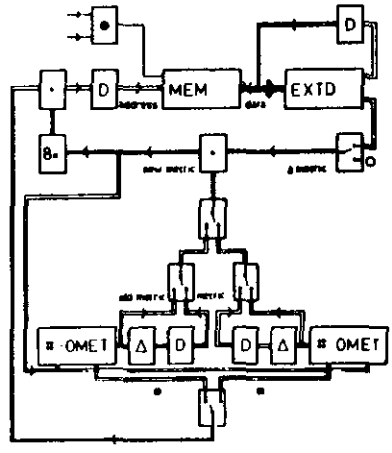
   -OMET module emits both the number #

   and the current lowest metric OMET.

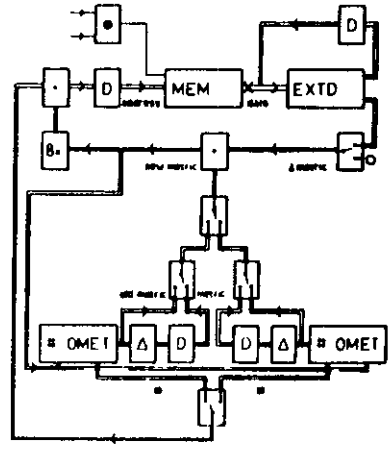
   From these the read address is computed.



2. Read a representative from the memory.



3. The EXTD module extends one representative into two, Both have to be stored.

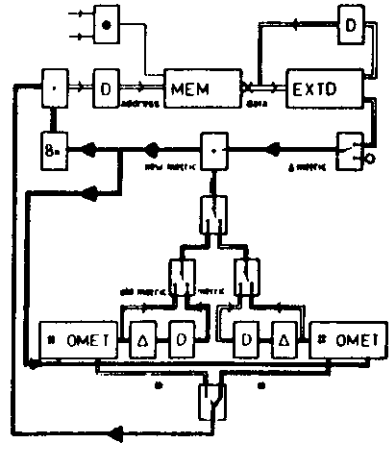


4. A store address is found from

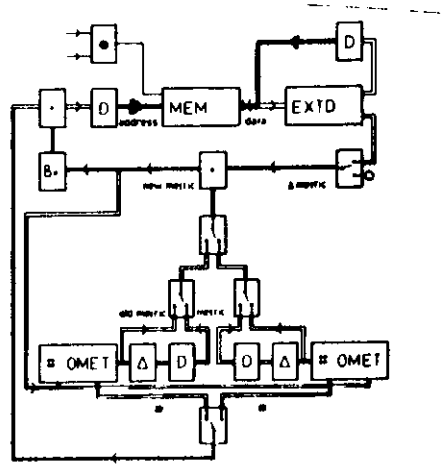
Δ METRIC (from EXTD module)
METRIC (from Δ module)
$\Rightarrow$ NMET

NMET is fed to the right # -OMET module to find the adjusted number # of stored representatives with a metric equal to NMET.

NMET
#
$\Rightarrow$ store address



5. The representative is stored.



If eight representatives have been read from the memory, extended and stored, the decoder input changes. At this time also the # -OMET modules interchange roles. By means of "pipe-lining", parallel processing according to Fig. 30 takes place.

```
1  2  3  4  5
      1  2  3  4  5
            1  2  3  4  5
                  1  2  3  4  5
```

Fig. 30. "Pipe-lining".

By this scheme, bus conflicts and double use of modules are avoided. To avoid the parallelogram like structure of Fig.30 which is nuissance to the control signal generation, we reset the # -OMET module, used to find the store addresses, to zero just before the first address of a cycle is to be found.

Conclusions

In this report we give two decoder implementations for convolutional codes, for which the class $L_{2,\nu,\nu-1}$ gives a significant gain in computational complexity. Both applications can be extended to the case of soft decisions. It is shown that for $p_{BSC} = 0.045$ the bit error probability $P_B \approx c.2^{-\log(m)}$. These results are in agreement with theoretical results as given in [11]. For small values of p_{BSC}, a very low decoder complexity can be obtained.

Acknowledgements.

The authors wish to thank the group of Prof. Schalkwijk for assistance and valuable discussions. The hardware implementation was carried out at the Philips Video pre-development group of Dr. U.Krause.

References

[1] Forney, Jr., G.D.
CONVOLUTIONAL CODES I: Algebraic structure.
IEEE Trans. Inf. Theory, Vol. IT-16(1970), p. 720-738.

[2] Viterbi, A.J.
CONVOLUTIONAL CODES AND THEIR PERFORMANCE IN COMMUNICATION SYSTEMS. IEEE Trans. Commun. Technol., Vol. COM-19(1971), p. 751-772.

[3] Fano, R.M.
A HEURISTIC DISCUSSION OF PROBABILISTIC DECODING.
IEEE Trans. Inf. Theory, Vol. IT-9(1963), p. 64-74.

[4] Massey, J.L.
VARIABLE-LENGTH CODES AND THE FANO METRIC.
IEEE Trans. Inf. Theory, Vol. IT-18(1972), p. 196-198.

[5] Vinck, A.J.
SYNDROME DECODING OF BINARY CONVOLUTIONAL CODES.
Ph.D. Thesis, Eindhoven University of Technology, 1980.

[6] Vinck, A.J., A.J.P. de Paepe and J.P.M. Schalkwijk
A CLASS OF BINARY RATE ONE-HALF CONVOLUTIONAL CODES THAT ALLOWS AN IMPROVED STACK DECODER.
IEEE Trans. Inf. Theory, Vol. IT-26(1980), p. 389-392.

[7] Massey, J.L.
LECTURE NOTES. Lehrgang Carl Cranz Gesellschaft (Germany), June 1975.

[8] Massey, J.L. and D.J. Costello, Jr.
NONSYSTEMATIC CONVOLUTIONAL CODES FOR SEQUENTIAL DECODING IN SPACE APPLICATIONS.
IEEE Trans. Commun. Technol., Vol. COM-19(1971), p. 806-813.

[9] Johannesson, R.
ROBUSTLY OPTIMAL RATE ONE-HALF BINARY CONVOLUTIONAL CODES.
IEEE Trans. Inf. Theory, Vol. IT-21(1975), p. 464-468.

[10] Jelinek, F.
PROBABILISTIC INFORMATION THEORY. New York: McGraw-Hill, 1968.
McGraw-Hill Series in System Sciences.

[11] Viterbi, A.J. and J.K. Omura
PRINCIPLES OF DIGITAL COMMUNICATION AND CODING.
New York: McGraw-Hill, 1979.
McGraw-Hill Series in Electrical Engineering.

EINDHOVEN UNIVERSITY OF TECHNOLOGY
THE NETHERLANDS
DEPARTMENT OF ELECTRICAL ENGINEERING

Reports:

93) Duin, C.A. van
DIPOLE SCATTERING OF ELECTROMAGNETIC WAVES PROPAGATION THROUGH A RAIN MEDIUM. TH-Report 79-E-93. 1979. ISBN 90-6144-093-9

94) Kuijper, A.H. de and L.K.J. Vandamme
CHARTS OF SPATIAL NOISE DISTRIBUTION IN PLANAR RESISTORS WITH FINITE CONTACTS. TH-Report 79-E-94. 1979. ISBN 90-6144-094-7

95) Hajdasinski, A.K. and A.A.H. Damen
REALIZATION OF THE MARKOV PARAMETER SEQUENCES USING THE SINGULAR VALUE DECOMPOSITION OF THE HANKEL MATRIX. TH-Report 79-E-95. 1979. ISBN 90-6144-095-5

96) Stefanov, B.
ELECTRON MOMENTUM TRANSFER CROSS-SECTION IN CESIUM AND RELATED CALCULATIONS OF THE LOCAL PARAMETERS OF Cs + Ar MHD PLASMAS. TH-Report 79-E-96. 1979. ISBN 90-6144-096-3

97) Worm, S.C.J.
RADIATION PATTERNS OF CIRCULAR APERTURES WITH PRESCRIBED SIDELOBE LEVELS. TH-Report 79-E-97. 1979. ISBN 90-6144-097-1

98) Kroezen, P.H.C.
A SERIES REPRESENTATION METHOD FOR THE FAR FIELD OF AN OFFSET REFLECTOR ANTENNA. TH-Report 79-E-98. 1979. ISBN 90-6144-098-X

99) Koonen, A.M.J.
ERROR PROBABILITY IN DIGITAL FIBER OPTIC COMMUNICATION SYSTEMS. TH-Report 79-E-99. 1979. ISBN 90-6144-099-8

100) Naidu, M.S.
STUDIES ON THE DECAY OF SURFACE CHARGES ON DIELECTRICS. TH-Report 79-E-100. 1979. ISBN 90-6144-100-5

101) Verstappen, H.L.
A SHAPED CYLINDRICAL DOUBLE-REFLECTOR SYSTEM FOR A BROADCAST-SATELLITE ANTENNA. TH-Report 79-E-101. 1979. ISBN 90-6144-101-3

102) Etten, W.C. van
THE THEORY OF NONLINEAR DISCRETE-TIME SYSTEMS AND ITS APPLICATION TO THE EQUALIZATION OF NONLINEAR DIGITAL COMMUNICATION CHANNELS. TH-Report 79-E-102. 1979. ISBN 90-6144-102-1

103) Roer, Th.G. van de
ANALYTICAL THEORY OF PUNCH-THROUGH DIODES. TH-Report 79-E-103. 1979. ISBN 90-6144-103-X

104) Herben, M.H.A.J.
DESIGNING A CONTOURED BEAM ANTENNA. TH-Report 79-E-104. 1979. ISBN 90-6144-104-8

EINDHOVEN UNIVERSITY OF TECHNOLOGY
THE NETHERLANDS
DEPARTMENT OF ELECTRICAL ENGINEERING

Reports:

105) Videc, M.F.
STRALINGSVERSCHIJNSELEN IN PLASMA'S EN BEWEGENDE MEDIA: Een geometrisch-optische en een golfzonebenadering.
TH-Report 80-E-105. 1980. ISBN 90-6144-105-6

106) Hajdasiński, A.K.
LINEAR MULTIVARIABLE SYSTEMS: Preliminary problems in mathematical description, modelling and identification.
TH-Report 80-E-106. 1980. ISBN 90-6144-106-4

107) Heuvel, W.M.C. van den
CURRENT CHOPPING IN SF_6.
TH-Report 80-E-107. 1980. ISBN 90-6144-107-2

108) Etten, W.C. van and T.M. Lammers
TRANSMISSION OF FM-MODULATED AUDIOSIGNALS IN THE 87.5 - 108 MHz BROADCAST BAND OVER A FIBER OPTIC SYSTEM.
TH-Report 80-E-108. 1980. ISBN 90-6144-108-0

109) Krause, J.C.
SHORT-CURRENT LIMITERS: Literature survey 1973-1979.
TH-Report 80-E-109. 1980. ISBN 90-6144-109-9

110) Matacz, J.S.
UNTERSUCHUNGEN AN GYRATORFILTERSCHALTUNGEN.
TH-Report 80-E-110. 1980. ISBN 90-6144-110-2

111) Otten, R.H.J.M.
STRUCTURED LAYOUT DESIGN.
TH-Report 80-E-111. 1980. ISBN 90-6144-111-0 (in preparation)

112) Worm, S.C.J.
OPTIMIZATION OF SOME APERTURE ANTENNA PERFORMANCE INDICES WITH AND WITHOUT PATTERN CONSTRAINTS.
TH-Report 80-E-112. 1980. ISBN 90-6144-112-9

113) Theeuwen, J.F.M. en J.A.G. Jess
EEN INTERACTIEF FUNCTIONEEL ONTWERPSYSTEEM VOOR ELEKTRONISCHE SCHAKELINGEN.
TH-Report 80-E-113. 1980. ISBN 90-6144-113-7

114) Lammers, T.M. en J.L. Manders
EEN DIGITAAL AUDIO-DISTRIBUTIESYSTEEM VOOR 31 STEREOKANALEN VIA GLASVEZEL.
TH-Report 80-E-114. 1980. ISBN 90-6144-114-5