Iterative Decoding of Binary Block and Convolutional Codes

Joachim Hagenauer, *Fellow, IEEE*, Elke Offer, and Lutz Papke

***Abstract*— Iterative decoding of two-dimensional systematic convolutional codes has been termed "turbo" (de)coding. Using log-likelihood algebra, we show that any decoder can be used which accepts soft inputs—including *a priori* values—and delivers soft outputs that can be split into three terms: the soft channel and *a priori* inputs, and the *extrinsic* value. The extrinsic value is used as an *a priori* value for the next iteration. Decoding algorithms in the log-likelihood domain are given not only for convolutional codes but also for any linear binary systematic block code. The iteration is controlled by a stop criterion derived from cross entropy, which results in a minimal number of iterations. Optimal and suboptimal decoders with reduced complexity are presented. Simulation results show that very simple component codes are sufficient, block codes are appropriate for high rates and convolutional codes for lower rates less than $2/3$. Any combination of block and convolutional component codes is possible. Several interleaving techniques are described. At a bit error rate (BER) of 10^{-4} the performance is slightly above or around the bounds given by the cutoff rate for reasonably simple block/convolutional component codes, interleaver sizes less than 1000 and for three to six iterations.**

***Index Terms*— Concatenated codes, product codes, iterative decoding, "soft-in/soft-out" decoder, "turbo" (de)coding.**

I. Introduction

SINCE the early days of information and coding theory the goal has always been to come close to the Shannon limit performance with a tolerable complexity. The results achieved so far show that it is relatively easy to operate at signal-to-noise ratios of E_b/N_0 above the value determined by the channel cutoff rate. For a rate $1/2$ code and soft decisions on a binary input additive white Gaussian noise (AWGN) channel the cutoff rate bound is at 2.5 dB, as opposed to the capacity limit which for rate $1/2$ is at 0.2 dB. It is generally held that between those two values of E_b/N_0 the task becomes very complex. Previously known methods of breaking this barrier were a) sequential decoding with the drawback of time and/or storage overflow and b) concatenated coding using Viterbi and Reed–Solomon decoders which achieve 1.6 dB at the cost of a large interleaver and feedback between two decoders [1].

Recently, interest has focused on iterative decoding of product or concatenated codes using "soft-in/soft-out" decoders with fairly simple component codes in an interleaved scheme. The basic idea is to break up decoding of a fairly complex and long code into steps while the transfer of probabilities or "soft" information between the decoding steps guarantees almost no loss of information. A flavor of the idea can be found in the work of Battail, e.g., [2]–[4]. Iterative decoding schemes with "soft-in/soft-out" decoders were proposed in [5]–[7]. In [6] an E_b/N_0 of 1.3 dB was achieved for the above mentioned channel with a three-dimensional code of moderate complexity. Impressive simulation results were presented in [8] achieving an E_b/N_0 of 0.7 dB, although with a huge interleaver of 64 500 bits, 18 iterations, and some *ad hoc* "fine-tuning" factors in the Bahl algorithm [9]. The novelty in [8] was the use of systematic feedback convolutional codes in the iterative scheme and the introduction of a "pseudo"-random interleaver (scrambler) between the two encoders. In the paper [10] some information-theory-based interpretation of iterative decoding is given.

The intention of this paper is to present the method of iterative decoding in a unified framework. We shall present several "soft-in/soft-out" algorithms which have the desired property that extrinsic information is used as *a priori* information in the next iteration step. We will show that any linear binary code in systematic form can be used as the component code and that "soft-in/soft-out" algorithms exist for these codes. The problem is the complexity; therefore, low-complexity algorithms such as the modified soft-output Viterbi algorithm (SOVA) will be presented [11], [12]. Cross-entropy introduced in [13] and [14] for iterative decoding will provide a useful criterion for stopping the iterations. Unfortunately, satisfying analytic results are not yet available. We shall present simulation results for convolutional and block codes.

II. Tools for Iterative Decoding of Binary Codes

A. Log-Likelihood Algebra

Let U be in GF(2) with the elements $\{+1, -1\}$, where $+1$ is the "null" element under the $\oplus$ addition. The log-likelihood ratio of a binary random variable U, $L_U(u)$, is defined as

$$L_U(u) = \log \frac{P_U(u=+1)}{P_U(u=-1)}. \tag{1}$$

Here $P_U(u)$ denotes the probability that the random variable U takes on the value u. The log-likelihood ratio $L_U(u)$ will be denoted as the "soft" value, or the L-value of the random

Manuscript received September 7, 1994; revised August 20, 1995.

J. Hagenauer is with the Technical University of Munich, D-80290 Munich, Germany.

E. Offer and L. Papke are with the Institute for Communications Technology, German Aerospace Research Establishment (DLR), Oberpfaffenhofen, D-82230 Wessling, P.O. Box 1116, Germany.

Publisher Item Identifier S 0018-9448(96)01474-5.

variable U. The sign of $L_U(u)$ is the hard decision and the magnitude $|L_U(u)|$ is the reliability of this decision. Unless stated otherwise, the logarithm is the natural logarithm.

If the binary random variable U is conditioned on a different random variable or vector Y, then we have a conditioned log-likelihood ratio $L_{U|Y}(u|y)$ with

$$\begin{aligned} L_{U|Y}(u|y) &= \log \frac{P_U(u=+1|y)}{P_U(u=-1|y)} \\ &= \log \frac{P_U(u=+1)}{P_U(u=-1)} + \log \frac{p_{Y|U}(y|u=+1)}{p_{Y|U}(y|u=-1)} \\ &= L_U(u) + L_{Y|U}(y|u). \end{aligned} \tag{2}$$

When there is no danger of confusion, we will henceforth skip the indices for the probabilities and the log-likelihood ratios. Notice that the joint log-likelihood $L(u,y)$ is equal to the conditioned log-likelihood $L(u|y)$ since the probability $P(y)$ term can be canceled out. Using the relations

$$\begin{aligned} P(u_1 \oplus u_2 = +1) &= P(u_1=+1) \cdot P(u_2=+1) \\ &\quad + (1 - P(u_1=+1)) \cdot (1 - P(u_2=+1)) \end{aligned} \tag{3}$$

with

$$P(u=+1) = \frac{e^{L(u)}}{1+e^{L(u)}} \tag{4}$$

it is easy to prove for statistically independent random variables U_1 and U_2

$$L(u_1 \oplus u_2) = \log \frac{1 + e^{L(u_1)} e^{L(u_2)}}{e^{L(u_1)} + e^{L(u_2)}} \tag{5}$$

$$\begin{aligned} &\approx \operatorname{sign}(L(u_1)) \cdot \operatorname{sign}(L(u_2)) \\ &\quad \cdot \min(|L(u_1)|, |L(u_2)|). \end{aligned} \tag{6}$$

From now on we will use a special algebra for the log-likelihood ratio values $L(u)$: We use the symbol $\boxplus$ as the notation for the addition defined by

$$L(u_1) \boxplus L(u_2) \triangleq L(u_1 \oplus u_2) \tag{7}$$

with the additional rules

$$L(u) \boxplus \infty = L(u) \qquad L(u) \boxplus -\infty = -L(u) \tag{8}$$

and

$$L(u) \boxplus 0 = 0. \tag{9}$$

By induction one can further prove that

$$\begin{aligned} \sum_{j=1}^{J}\!\boxplus\, L(u_j) &\triangleq L\left(\sum_{j=1}^{J}\!\oplus\, u_j\right) \\ &= \log \frac{\prod_{j=1}^{J}(e^{L(u_j)}+1) + \prod_{j=1}^{J}(e^{L(u_j)}-1)}{\prod_{j=1}^{J}(e^{L(u_j)}+1) - \prod_{j=1}^{J}(e^{L(u_j)}-1)} \end{aligned} \tag{10}$$

is true. Using the relation $\tanh(u/2) = (e^u - 1)/(e^u + 1)$we obtain [3]

$$\begin{aligned} \sum_{j=1}^{J}\!\boxplus\, L(u_j) &= \log \frac{1 + \prod_{j=1}^{J} \tanh(L(u_j)/2)}{1 - \prod_{j=1}^{J} \tanh(L(u_j)/2)} \\ &= 2\,\operatorname{artanh}\left(\prod_{j=1}^{J} \tanh(L(u_j)/2)\right) \end{aligned} \tag{11}$$

and finally approximate it as in (6) by

$$\begin{aligned} \sum_{j=1}^{J}\!\boxplus\, L(u_j) &= L\left(\sum_{j=1}^{J}\!\oplus\, u_j\right) \\ &\approx \left(\prod_{j=1}^{J} \operatorname{sign}(L(u_j))\right) \cdot \min_{j=1\cdots J} |L(u_j)|. \end{aligned} \tag{12}$$

The reliability of the sum $\boxplus$ is therefore determined by the smallest reliability of the terms. From (11) we get the symmetrical relation

$$\tanh\left(\frac{1}{2} \sum_{j=1}^{J}\!\boxplus\, L(u_j)\right) = \prod_{j=1}^{J} \tanh(L(u_j)/2) \tag{13}$$

to be used in the Appendix.

B. Soft Channel Outputs

Now, we will define more clearly what is meant by the "soft values" of a channel. If we encode the binary value u having a soft value $L(u)$ then we create coded bits x with soft values $L(x)$. For an (N,K)-systematic code, K of the bits x are equal to the information bits u. After transmission over a binary symmetric channel (BSC) or a Gaussian/fading channel we can calculate the log-likelihood ratio of x conditioned on the matched filter output y

$$\begin{aligned} L(x|y) &= \log \frac{P(x=+1|y)}{P(x=-1|y)} \\ &= \log\left(\frac{p(y|x=+1)}{p(y|x=-1)} \cdot \frac{P(x=+1)}{P(x=-1)}\right). \end{aligned} \tag{14}$$

With our notation we obtain

$$\begin{aligned} L(x|y) &= \log \frac{\exp(-\frac{E_s}{N_0}(y-a)^2)}{\exp(-\frac{E_s}{N_0}(y+a)^2)} + \log \frac{P(x=+1)}{P(x=-1)} \\ &= L_c \cdot y + L(x) \end{aligned} \tag{15}$$

with $L_c = 4a \cdot E_s/N_0$. For a fading channel, a denotes the fading amplitude whereas for a Gaussian channel we set $a = 1$. For a BSC, L_c is the log-likelihood ratio of the crossover probabilities P_0, where $L_c = \log((1-P_0)/P_0)$. L_c is called the reliability value of the channel.

We further note that for statistically independent transmission, as in dual diversity or with a repetition code

$$L(x|y_1, y_2) = L_{c_1} y_1 + L_{c_2} y_2 + L(x). \tag{16}$$

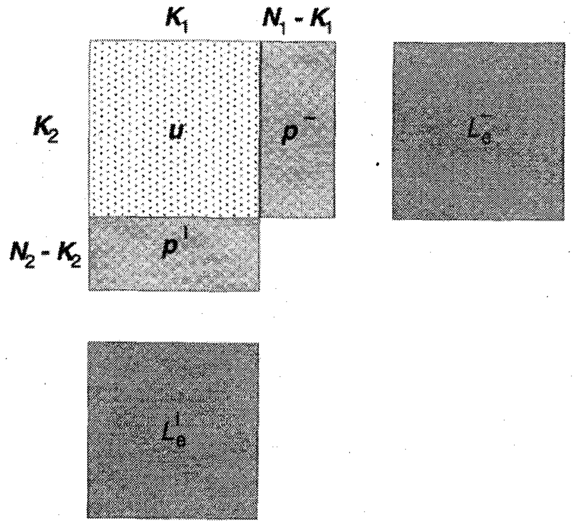

Fig. 1. Iterative decoding scheme for two-dimensional codes.

For the rest of the paper we assume a channel with constant reliability denoted by L_c. In the general case (fading, etc.) L_c is time-variant and would have the same additional index as y.

C. Principle of Iterative Decoding Algorithms

We show the principle of iterative decoding in the two-dimensional case [5], [8]. The $K_1 \cdot K_2$ information bits $\boldsymbol{u}$ are ordered in a rectangular matrix as shown in Fig. 1. Attached to it are the parity bits $\boldsymbol{p}^-$ and $\boldsymbol{p}^|$ of the two systematic codes C^- and $C^|$. The received values at the matched filter output are denoted by y and $L_c \cdot y$ which are available to the decoder for all coded bits.

We will first use a simple example to demonstrate the main ideas. Then we will proceed with the general case for block codes and for convolutional codes and conclude with some generalizations.

1) Tutorial Example with the $(3, 2, 2)$ *Single Parity Check Code as Component Code:* Let us encode four information bits by two $(3, 2, 2)$ single parity check codes with elements $\{+1, -1\}$ in GF(2) as shown in Fig. 2(b) and let us assume we have received the values $L_c \cdot y$ shown in Fig. 2(c). No *a priori* information is yet available. Let us start with horizontal decoding: The information for bit u_{11} is received twice: Directly via u_{11} and indirectly via $u_{12} \oplus p_1^-$. Since u_{12} and p_1^- are transmitted statistically independent we have for their L-value

$$L(u_{12} \oplus p_1^-) = L(u_{12}) \boxplus L(p_1^-) = 1.5 \boxplus 1.0 \approx 1.0.$$

This indirect information about u_{11} is called the extrinsic value and is stored in Fig. 2(d). For u_{12} we obtain by the same argument a horizontal extrinsic value of $0.5 \boxplus 1.0 \approx 0.5$ and so on for the second row. When the horizontal extrinsic table is filled we start vertical decoding using these L_e^- as *a priori* values for vertical decoding. This means that after

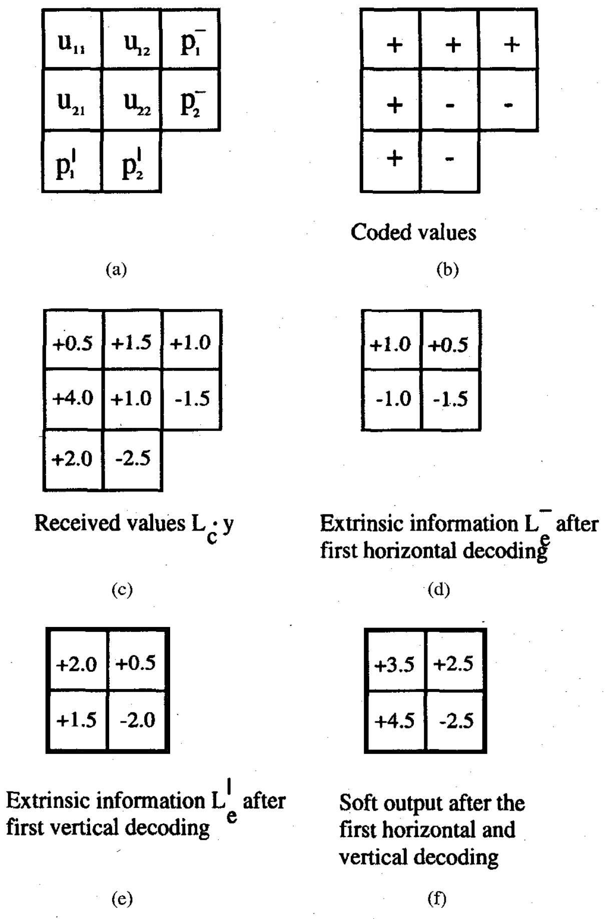
u_{11}
u_{12}
p_1^-
u_{21}
u_{22}
p_2^-
$p_1^|$
$p_2^|$
(a)
+ + +
+ - -
+ -
Coded values
(b)
+0.5 +1.5 +1.0
+4.0 +1.0 -1.5
+2.0 -2.5
Received values $L_c \cdot y$
(c)
+1.0 +0.5
-1.0 -1.5
Extrinsic information L_e^- after first horizontal decoding
(d)
+2.0 +0.5
+1.5 -2.0
Extrinsic information $L_e^|$ after first vertical decoding
(e)
+3.5 +2.5
+4.5 -2.5
Soft output after the first horizontal and vertical decoding
(f)

Fig. 2. Example for iterative decoding of a rate-1/2 code using two rate-2/3 single parity check codes.

vertical decoding of u_{11} we have the following three L-values available for u_{11}:

- the received direct value $+0.5$,
- the *a priori* value L_e^- from horizontal decoding $+1.0$ and
- the vertical extrinsic value $L_e^|$ using all the available information on $u_{21} \oplus p_1^|$, namely, $(4.0 + (-1.0)) \boxplus 2.0 \approx 2.0$.

The vertical extrinsic value is stored in the table of Fig. 2(e). For u_{21} it amounts to $(0.5 + 1.0) \boxplus 2.0 \approx 1.5$, for u_{12} to $(1.0 + (-1.5)) \boxplus (-2.5) \approx 0.5$, and for u_{22} to $(1.5 + 0.5) \boxplus (-2.5) \approx -2.0$. If we were to stop the iterations here we would obtain as soft output after the vertical iteration

$$L(\hat{u}) = L_c \cdot y + L_e^- + L_e^| \tag{17}$$

shown in Fig. 2(f). The addition in (17) is justified from (16) because up to now the three terms in (17) are statistically independent. We could now continue with another round of horizontal decoding using the respective $L_e^|$ as *a priori* information. However, now we encounter statistical dependencies. Anyway, in our example we have already correctly decoded with good reliabilities $|L(\hat{u})|$. The desired statistical independence is one of the reasons why we are not using a full

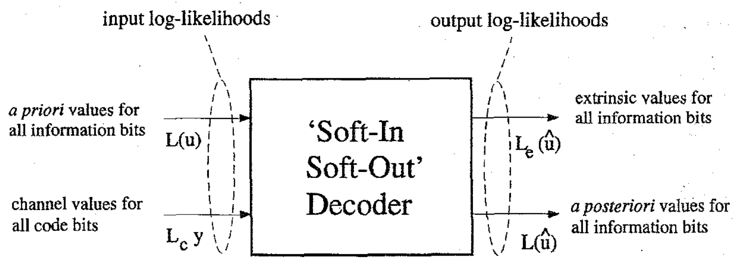
input log-likelihoods
output log-likelihoods
a priori values for all information bits
L(u)
channel values for all code bits
L_c y
'Soft-In Soft-Out' Decoder
extrinsic values for all information bits
$L_e(\hat{u})$
a posteriori values for all information bits
$L(\hat{u})$

Fig. 3. "Soft-in/soft-out" decoder.

product code. Note that we are determining extrinsic values to be used as *a priori* values only for information bits and not for parity bits, because codeword probabilities are determined from *a priori* probabilities of information bits only. This will become clear in Section III-C.

2) General Setup with Block Codes as Component Codes: We can use any combination of systematic block codes for encoding the $K_1 \cdot K_2$ information bits in the horizontal or vertical direction. One example might be

Horizontally: K_2 code words of a (K_1, N_1) block code C^- with rate $R_1 = K_1/N_1$.
Vertically: K_1 code words of a (K_2, N_2) block code $C^|$ with rate $R_2 = K_2/N_2$.

If we mean either the vertical or the horizontal code, we drop the indices 1 and 2. The total rate of the two-dimensional code will be

$$R = \frac{1}{1 + \frac{(N_2-K_2)K_1}{K_1K_2} + \frac{(N_1-K_1)K_2}{K_1K_2}} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} - 1}. \quad (18)$$

Each row or column of the information matrix forms an information sequence $\boldsymbol{u}$ to be encoded into a codeword

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_N) = (u_1, \cdots, u_K, p_1, \cdots, p_{N-K})$$

where $\boldsymbol{x} \in C^-$ or $\boldsymbol{x} \in C^|$, respectively.

In the remainder of this section we shall omit the indices in u and y for the sake of brevity. Assume we have a "soft-in/soft-out" decoder available as shown in Fig. 3 for decoding the component codes. The output of the "symbol-by-symbol"' maximum *a posteriori* (MAP) decoder is defined as the *a posteriori* log-likelihood ratio for a transmitted "+1" and a transmitted "−1" in the information sequence

$$L(\hat{u}) \triangleq L(u|\boldsymbol{y}) = \log \frac{P(u=+1|\boldsymbol{y})}{P(u=-1|\boldsymbol{y})}. \quad (19)$$

Such a decoder uses *a priori* values $L(u)$ for all information bits u, if available, and channel values $L_c \cdot y$ for all coded bits. It also delivers soft outputs $L(\hat{u})$ on all information bits and an *extrinsic* information $L_e(\hat{u})$ which contains the soft output information from all the other coded bits in the code sequence and is not influenced by the $L(u)$ and $L_c \cdot y$ values of the current bit. For systematic codes, the soft output for the information bit u will be represented in Section III in three additive terms

$$L(\hat{u}) = L_c \cdot y + L(u) + L_e(\hat{u}). \quad (20)$$

This means we have three independent estimates for the log-likelihood ratio of the information bits: The channel values $L_c \cdot y$, the *a priori* values $L(u)$ and the values $L_e(\hat{u})$ by a third independent estimator utilizing the code constraint. Assume equally likely information bits: Then we do not have any *a priori* information available for the first iteration, thus we initialize $L(u) = 0$. Decoding of the horizontal code C^- starts using the corresponding $L_c \cdot y$ for the information part and for the horizontal parity part. The extrinsic information $L_e^-(\hat{u})$ of the horizontal code C^- on the information bit u is from (20)

$$L_e^-(\hat{u}) = L^-(\hat{u}) - L_c \cdot y. \quad (21)$$

This independent estimate on u is now used as the *a priori* value for decoding code $C^|$ vertically to obtain

$$L_e^|(\hat{u}) = L^|(\hat{u}) - \left(L_c \cdot y + L_e^-(\hat{u})\right). \quad (22)$$

This vertical extrinsic information will be used as new *a priori* value in the subsequent decoding of code C^- in the next iteration step. Note that for the first horizontal and the first vertical iteration the L-values are statistically independent, but since later on they will use the same information indirectly, they will become more and more correlated and finally the improvement through the iterations will be marginal. Of course, for the final decision (or soft output) after the last vertical iteration we combine the last two extrinsic pieces of information with the received values to obtain

$$L(\hat{u}) = L_c \cdot y + L_e^-(\hat{u}) + L_e^|(\hat{u}) \quad (23)$$

which, using (22), is identical to $L^|(\hat{u})$. The whole procedure is shown in Fig. 4.

3) General Setup with Convolutional Codes as Component Codes: Convolutional codes are used with a systematic feedback realization of the encoder. If the generator matrix of a rate $1/n$ encoder is

$$G(D) = (g_0(D) \;\; g_1(D) \cdots g_{n-1}(D))$$

the feedback encoder will be

$$G_{\text{sys}}(D) = \left(1 \;\; \frac{g_1(D)}{g_0(D)} \cdots \frac{g_{n-1}(D)}{g_0(D)}\right). \quad (24)$$

We will later use the generator polynomials $g_0(D) = 1 + D + D^2$, $g_1(D) = 1 + D^2$, and $g_0(D) = 1 + D^3 + D^4$, $g_1(D) = 1 + D + D^2 + D^4$ for the rate-1/2 convolutional code with memory $m = 2$ and memory $m = 4$, respectively. Fig. 5

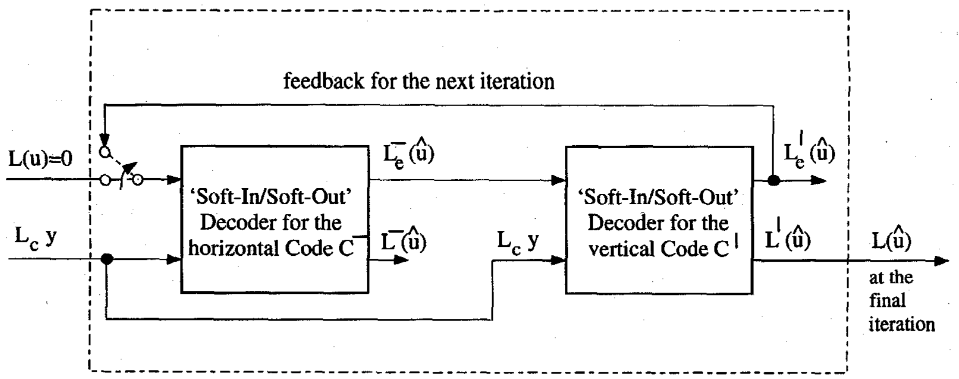
feedback for the next iteration
L(u)=0
L_c y
'Soft-In/Soft-Out' Decoder for the horizontal Code C^-
$L_e^-(\hat{u})$
$L^-(\hat{u})$
L_c y
'Soft-In/Soft-Out' Decoder for the vertical Code $C^|$
$L_e^|(\hat{u})$
$L^|(\hat{u})$
$L(\hat{u})$
at the final iteration

Fig. 4. Iterative decoding procedure with two "soft-in/soft-out" decoders with initial $L(u) = 0$, i.e., equally likely source (information) bits.

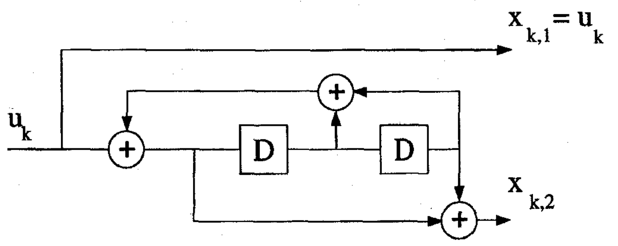

Fig. 5. Realization of the systematic convolutional encoder with feedback for the rate-1/2 code with memory 2. The generator polynomials are $g_0(D) = 1 + D + D^2$ and $g_1(D) = 1 + D^2$.

shows a realization of the convolutional encoder with feedback for the 4-state code. The parity check bits are punctured to achieve the desired higher rate $\tilde{k}/\tilde{n}$. Now we use as component codes

Horizontally:	A code sequence of a convolutional code of rate $R_1 = \tilde{k}_1/\tilde{n}_1$. This code is punctured from a rate $1/n_1$ mother code which has memory m_1 and a binary trellis with 2^{m_1} states. We assume that K_1 and N_1 are multiples of $\tilde{k}_1$ and $\tilde{n}_1$.
Vertically:	A code sequence of a convolutional code of rate $R_2 = \tilde{k}_2/\tilde{n}_2$, punctured from a rate $1/n_2$ mother code with 2^{m_2} states. Again K_2 and N_2 are chosen as multiples of $\tilde{k}_2$ and $\tilde{n}_2$.

Using a convolutional code, the $K_1 \cdot K_2$ information bits $\boldsymbol{u}$ are first encoded into the systematic code sequence

$$\boldsymbol{x} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k, \cdots, \boldsymbol{x}_{K_1 \cdot K_2})$$

with

$$\boldsymbol{x}_k = (x_{k,1}, x_{k,2}, \cdots, x_{k,n})^T = (u_k, p_{k,1}, \cdots, p_{k,n-1})^T \quad (25)$$

where $\boldsymbol{x} \in C^-$ or $\boldsymbol{x} \in C^|$, respectively. Some of the parity bits

$$x_{k,\nu} = p_{k,\nu-1}, \quad 1 \leq k \leq K_1 \cdot K_2, \ 2 \leq \nu \leq n$$

in the code sequence might be punctured according to the puncturing rule. If the information bit u_k is transmitted, $1 \leq k \leq K_1 \cdot K_2$, we receive the value $L_c \cdot y_{k,1}$. The respective values for the nonpunctured parities are $x_{k,\nu}$ and $L_c \cdot y_{k,\nu}$. Note that we use k as a running index either for the information bits, the coded bits, or the channel values.

The problem of the termination of the convolutional code can be solved by terminating code C^- by m_1 known bits and leaving code $C^|$ open [15].

4) Generalizations: Several generalizations of the decoding schemes in Sections II-C2 and II-C3 are possible.

a) Combinations of block and convolutional codes.
b) The extension to more than two dimensions is obvious and has been investigated in [5] and [6].
c) Although "soft-in/soft-out" decoding is in principle also possible with codes in nonsystematic form, we only will use codes with systematic encoders for the following reasons:
    - Codes in their systematic and nonsystematic form give equivalent codes.
    - It is well known that systematic convolutional feed-forward encoders produce less powerful codes and therefore they are not considered here. Convolutional codes in their systematic feedback form are equivalent to the nonsystematic form in distance and nearest neighbor path properties. However, the BER at low signal-to-noise ratio (SNR) is slightly better.
    - A nonsystematic two-dimensional implementation would force us to transmit the encoded information part twice, reducing the overall code rate dramatically.
d) In the rare cases where we have outside information that the source (information) bits are not equally likely, we have to add this source *a priori* information $L_s(u)$ to all the *a priori* values in the iterations.
e) The interleaver need not be in vertical and horizontal block form. Any "pseudo"-random permutation of the information bits for the second encoding is possible and might result in a better BER [8], [16], and [17].

D. Convergence Properties of Iterative Decoding via Cross-Entropy

Battail [13] and Moher [14] have shown that cross-entropy is a useful criterion for iterative decoding. We will show how cross-entropy transforms into our notation and that it is a useful stop criterion for an iterative algorithm.

Let the soft output have the structure as described in Section II-C, (23). We then have two *a posteriori* distributions of subsequent decoding operations. The cross-entropy of two distributions $\boldsymbol{P}(\hat{\boldsymbol{u}})$ and $\boldsymbol{Q}(\hat{\boldsymbol{u}})$ is defined as

$$\boldsymbol{E_P}\left\{\log\frac{\boldsymbol{P}(\hat{\boldsymbol{u}})}{\boldsymbol{Q}(\hat{\boldsymbol{u}})}\right\} \tag{26}$$

and is a measure of the difference ("closeness") of two distributions. Here $\boldsymbol{E_P}$ denotes the expectation operator over the distribution $\boldsymbol{P}(\hat{\boldsymbol{u}})$. Assuming statistical independence, we obtain

$$\log\frac{\boldsymbol{P}(\hat{\boldsymbol{u}})}{\boldsymbol{Q}(\hat{\boldsymbol{u}})} = \sum_k \log\frac{P(\hat{u}_k)}{Q(\hat{u}_k)}. \tag{27}$$

Now, let us look at two subsequent iterations $(i-1)$ and (i), where one iteration consists of the decoding in the "horizontal" and the "vertical" direction. We define

$$L_Q^{(i)}(\hat{u}_k) = L_c \cdot y_k + L_e^{-(i-1)}(\hat{u}_k) + L_e^{|(i)}(\hat{u}_k) \tag{28}$$

$$L_P^{(i)}(\hat{u}_k) = L_c \cdot y_k + L_e^{|(i)}(\hat{u}_k) + L_e^{-(i)}(\hat{u}_k) \tag{29}$$

and therefore the difference in the soft outputs equals

$$\begin{aligned} L_P^{(i)}(\hat{u}_k) - L_Q^{(i)}(\hat{u}_k) &= L_e^{-(i)}(\hat{u}_k) - L_e^{-(i-1)}(\hat{u}_k) \\ &= \Delta L_e^{-(i)}(\hat{u}_k). \end{aligned} \tag{30}$$

Using the inverse of (1)

$$\hat{u}_k^{(i)} = \operatorname{sign}(L_P^{(i)}(\hat{u}_k))$$

and

$$|L_P^{(i)}(\hat{u}_k)| = \hat{u}_k^{(i)} \cdot L_P^{(i)}(\hat{u}_k)$$

it is straightforward to show that

$$\begin{aligned} E_P\left\{\log\frac{P(\hat{u}_k)}{Q(\hat{u}_k)}\right\} &= P(\hat{u}_k=+1)\log\frac{P(\hat{u}_k=+1)}{Q(\hat{u}_k=+1)} \\ &\quad + P(\hat{u}_k=-1)\log\frac{P(\hat{u}_k=-1)}{Q(\hat{u}_k=-1)} \\ &= -\Delta L_e^{-(i)}\frac{1}{1+\exp(L_P^{(i)}(\hat{u}_k))} \\ &\quad + \log\frac{1+\exp(-L_Q^{(i)}(\hat{u}_k))}{1+\exp(-L_P^{(i)}(\hat{u}_k))} \\ &\approx -\hat{u}_k^{(i)}\Delta L_e^{-(i)}(\hat{u}_k)\frac{1}{1+\exp(|L_P^{(i)}(\hat{u}_k)|)} \\ &\quad + \log\frac{1+\exp(-|L_Q^{(i)}(\hat{u}_k)|)}{1+\exp(-|L_P^{(i)}(\hat{u}_k)|)}. \end{aligned} \tag{31}$$

The last approximation is valid, when the decisions do not change anymore, i.e., when

$$\operatorname{sign}(L_P^{(i)}(\hat{u}_k)) = \operatorname{sign}(L_Q^{(i)}(\hat{u}_k)) = \hat{u}_k^{(i)}.$$

Further, if the reliabilities are large enough, we have with $\log(1+x) \approx x$

$$\begin{aligned} E_P\left\{\log\frac{P(\hat{u}_k)}{Q(\hat{u}_k)}\right\} &\approx \exp(-|L_Q^{(i)}(\hat{u}_k)|) \\ &\quad \cdot \left(1-\exp\left(-\hat{u}_k^{(i)}\Delta L_e^{-(i)}(\hat{u}_k)\right)\right. \\ &\quad \left.\cdot \left(1+\hat{u}_k^{(i)}\Delta L_e^{-(i)}(\hat{u}_k)\right)\right). \end{aligned} \tag{32}$$

As long as $\Delta L_e^{-(i)}$ has the same sign as $\hat{u}_k^{(i)}$ and a magnitude smaller than 1, we take the first two terms of the series expansion of $\exp(x)$ and further obtain

$$\boldsymbol{E_P}\left\{\log\frac{\boldsymbol{P}(\hat{\boldsymbol{u}})}{\boldsymbol{Q}(\hat{\boldsymbol{u}})}\right\} \approx \sum_k \frac{\left|\Delta L_e^{-(i)}(\hat{u}_k)\right|^2}{\exp(|L_Q^{(i)}(\hat{u}_k)|)}. \tag{33}$$

Of course, the assumption of statistical independence between likelihood values is not exactly true after some iterations. Nevertheless, we could use the criterion

$$T(i) = \sum_k \frac{\left|\Delta L_e^{-(i)}(\hat{u}_k)\right|^2}{\exp(|L_Q^{(i)}(\hat{u}_k)|)} < \text{threshold} \tag{34}$$

as a stop criterion for the iterations. Simulation results have shown that $T(i)$ drops by a factor of 10^{-2} to 10^{-4} once no more errors will be corrected and a threshold value of $T(1)\cdot 10^{-3}$ is appropriate to stop the iterations. The benefit of this stop criterion will be shown in Section IV-A in Table II.

III. Optimal and Suboptimal Algorithms

A. *"Symbol-by-Symbol" Maximum A Posteriori Probability (MAP) Decoding Rule for Systematic Convolutional Codes in Feedback Form with a Binary Trellis*

The MAP algorithm for trellis codes was proposed simultaneously by Bahl, Cocke, Jelinek, and Raviv in [18] (and later in [9]) and by McAdam, Welch, and Weber in [19]. In [8] the algorithm was adapted to systematic convolutional codes. Here we will show how the MAP decoder uses log-likelihood values and that its output has the general structure given in (20). In [16] and [20] the structure of such decoders was illuminated and simulation results for the Bahl algorithm including optimized interleavers were presented.

The trellis of a binary feedback convolutional encoder has the structure shown in Fig. 6. Let S_k be the encoder state at time k. The bit u_k is associated with the transition from time $k-1$ to time k. The trellis states at level $k-1$ and at level k are indexed by the integer s' and s, respectively. The goal of the MAP algorithm is to provide us with

$$L(\hat{u}_k) = \log\frac{P(u_k=+1|\boldsymbol{y})}{P(u_k=-1|\boldsymbol{y})} = \log\frac{\sum\limits_{\substack{(s',s)\\u_k=+1}} p(s',s,\boldsymbol{y})}{\sum\limits_{\substack{(s',s)\\u_k=-1}} p(s',s,\boldsymbol{y})}. \tag{35}$$

The index pair s' and s determines the information bit u_k and the coded bits $x_{k,\nu}$, for $\nu = 2,\cdots,n$. The sum of the joint probabilities $p(s',s,\boldsymbol{y})$ in the numerator or in the denominator of (35) is taken over all existing transitions from state s' to state s labeled with the information bit $u_k = +1$ or with $u_k = -1$, respectively. Assuming a memoryless transmission channel, the joint probability $p(s', s, \boldsymbol{y})$ can be written as the product of three independent probabilities [9]

$$\begin{aligned} p(s', s, \boldsymbol{y}) &= p(s', \boldsymbol{y}_{j<k}) \cdot p(s, \boldsymbol{y}_k | s') \cdot p(\boldsymbol{y}_{j>k} | s) \\ &= \underbrace{p(s', \boldsymbol{y}_{j<k})}_{} \cdot \underbrace{P(s|s') \cdot p(\boldsymbol{y}_k | s', s)}_{} \cdot \underbrace{p(\boldsymbol{y}_{j>k} | s)}_{} \\ &= \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s). \end{aligned} \quad (36)$$

Here $\boldsymbol{y}_{j<k}$ denotes the sequence of received symbols $\boldsymbol{y}_j$ from the beginning of the trellis up to time $k-1$ and $\boldsymbol{y}_{j>k}$ is the corresponding sequence from time $k+1$ up to the end of the trellis. The forward recursion of the MAP algorithm yields

$$\alpha_k(s) = \sum_{s'} \gamma_k(s', s) \cdot \alpha_{k-1}(s'). \quad (37)$$

The backward recursion yields

$$\beta_{k-1}(s') = \sum_{s} \gamma_k(s', s) \cdot \beta_k(s). \quad (38)$$

In order to perform the optimum "symbol-by-symbol" MAP rule, the trellis has to be of finite duration. We assume that at the start and at the end of the observed sequence all paths merge at the zero state. Then the forward and backward recursion are initialized with $\alpha_{\text{start}}(0) = 1$ and $\beta_{\text{end}}(0) = 1$. Whenever a transition between s' and s exists the branch transition probabilities are given by

$$\gamma_k(s', s) = p(\boldsymbol{y}_k | u_k) \cdot P(u_k). \quad (39)$$

Using the log-likelihoods, the *a priori* probability $P(u_k)$ can be expressed as

$$\begin{aligned} P(u_k = \pm 1) &= \frac{e^{\pm L(u_k)}}{1 + e^{\pm L(u_k)}} = \left(\frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}} \right) \cdot e^{L(u_k) u_k / 2} \\ &= A_k \cdot e^{L(u_k) u_k / 2} \end{aligned} \quad (40)$$

and, in a similar way, the conditioned probability $p(\boldsymbol{y}_k | u_k)$ for systematic convolutional codes can be written as

$$p(\boldsymbol{y}_k | u_k) = B_k \cdot \exp\left(\frac{1}{2} L_c y_{k,1} u_k + \frac{1}{2} \sum_{\nu=2}^{n} L_c y_{k,\nu} x_{k,\nu} \right). \quad (41)$$

Keep in mind that some of the coded bits might be punctured before transmission, in which case the sum in (41) is only over those indices ν corresponding to nonpunctured coded bits. The terms A_k and B_k in (40) and (41) are equal for all transitions from level $k-1$ to level k and hence will cancel out in the ratio of (35). Therefore, the branch transition operation to be used in (37) and (38) reduces to the expression

$$\exp\left(\frac{1}{2} u_k (L_c y_{k,1} + L(u_k)) \right) \cdot \gamma_k^{(e)}(s', s) \quad (42)$$

with

$$\gamma_k^{(e)}(s', s) = \exp\left(\frac{1}{2} \sum_{\nu=2}^{n} L_c y_{k,\nu} x_{k,\nu} \right). \quad (43)$$

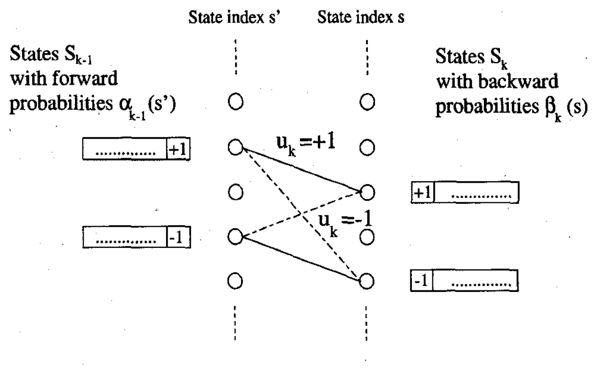
State index s'
State index s
States S_{k-1} with forward probabilities $\alpha_{k-1}(s')$
States S_k with backward probabilities $\beta_k(s)$
u_k=+1
u_k=-1
+1
-1
+1
-1
At the transistions the labels are $x_{k,1} = u_k, x_{k,2}, \ldots, x_{k,n}$

Fig. 6. Trellis structure of systematic convolutional codes with feedback encoders.

Since the first exponential function in (42) is common in all terms in the sums of (35), we divide all terms by those and obtain

$$L(\hat{u}_k) = L_c y_{k,1} + L(u_k) + \log \frac{\sum\limits_{\substack{(s',s) \\ u_k=+1}} \gamma_k^{(e)}(s', s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum\limits_{\substack{(s',s) \\ u_k=-1}} \gamma_k^{(e)}(s', s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}. \quad (44)$$

Thus we have shown that the MAP algorithm for systematic codes has the structure of (20). We can avoid calculating actual probabilities by using the logarithm of probabilities and the approximation $\log(e^{L_1} + e^{L_2}) \approx \max(L_1, L_2)$. Then this algorithm works with $\log \alpha_k(s)$, $\log \beta_{k-1}(s')$, and $\log \gamma_k(s', s)$ and the summations in (37), (38), and (44) are replaced by the corresponding maximizations. For the remainder of the paper we will refer to this suboptimal realization of the "symbol-by-symbol" MAP rule as the Log-MAP rule realization. Investigations have shown that the performance of the Log-MAP algorithm is close to the optimal "symbol-by-symbol" MAP algorithm, in particular when the above approximation is improved by adding a correction term to $\max(L_1, L_2)$ having eight possible values [21].

B. The "Soft-In/Soft-Out" Viterbi Algorithm (SOVA) for Systematic Convolutional Codes in Feedback Form with a Binary Trellis

The Viterbi algorithm (VA) in its MAP form is described in [22]. It searches for the ith-state sequence $\boldsymbol{S}^{(i)}$ and thus the desired information sequence $\boldsymbol{u}^{(i)}$ by maximizing over i the *a posteriori* probability

$$P(\boldsymbol{S}^{(i)} | \boldsymbol{y}) = p(\boldsymbol{y} | \boldsymbol{S}^{(i)}) \frac{P(\boldsymbol{S}^{(i)})}{p(\boldsymbol{y})}. \quad (45)$$

Since $\boldsymbol{y}$ is fixed we can equivalently maximize

$$p(\boldsymbol{y} | \boldsymbol{S}^{(i)}) P(\boldsymbol{S}^{(i)}). \quad (46)$$

This maximization is realized in the code trellis, when for each state s and each time k, the path with the largest probability

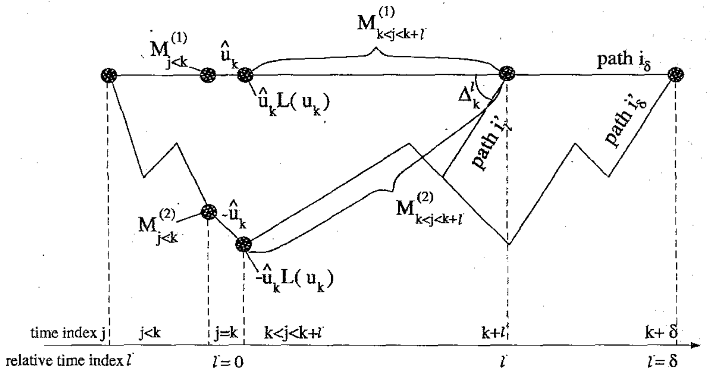

Fig. 7. Example for the derivation of Δ^l_k.

$p(\boldsymbol{S}^{(i)}_{j\le k}, \boldsymbol{y}_{j\le k})$ is selected. This probability can be calculated by multiplying the branch transition probabilities associated to path i. They are $\gamma_j(s'^{(i)}, s^{(i)})$ for $1 \le j \le k$ and defined in (39). The maximum is not changed if we take the logarithm, and hence we perform the same metric computation as described for the forward recursion of the Log-MAP algorithm in Section III-A. The values $\log A_k$ and $\log B_k$ from (40) and (41) are additive and the same for all paths i and therefore are irrelevant for the maximization. As already mentioned above, we assume hereby a memoryless transmission channel and statistical independence of the relevant u within the observation window of the VA. For the metric of the ith path at time k we obtain

$$M_k(s^{(i)}) = M_{k-1}(s'^{(i)}) + \frac{1}{2}L(u_k)u^{(i)}_k + \frac{1}{2}\sum_{\nu=1}^{n} L_c y_{k,\nu} x^{(i)}_{k,\nu}. \tag{47}$$

Here $s^{(i)}$ denotes the state of the path i at time k, $u^{(i)}_k$ is the information bit, and $x^{(i)}_{k,\nu}$ are the coded bits of path i at time k. For systematic codes we further have

$$M_k(s^{(i)}) = M_{k-1}(s'^{(i)}) + \frac{1}{2}L_c y_{k,1} u^{(i)}_k + \frac{1}{2}L(u_k)u^{(i)}_k + \frac{1}{2}\sum_{\nu=2}^{n} L_c y_{k,\nu} x^{(i)}_{k,\nu}. \tag{48}$$

Again the sum is over the indices ν with nonpunctured coded bits. A different derivation of the path metric (47) can be found in [12].

This slight modification of the metric of the VA in (47) and in (48) incorporates the *a priori* information about the probability of the information bits. Forney [22] already mentioned the possibility of using *a priori* values in his paper, but did not give any use or application for it. If the channel is very good, $|L_c \cdot y|$ will be larger than $|L(u)|$, and decoding relies on the received channel values. If the channel is bad, as during a deep fade, decoding relies on the *a priori* information $L(u)$. In iterative decoding this is the extrinsic value from the previous decoding step.

Note that at time k, the joint probability of the path i and of the received sequence $\boldsymbol{y}_{j\le k}$ and the metric in (48) are related by

$$p(path\ i, \boldsymbol{y}_{j\le k}) = p(\boldsymbol{S}^{(i)}_{j\le k}, \boldsymbol{y}_{j\le k}) = \left(\prod_{j=1}^{k} A_j \cdot B_j\right) \cdot e^{M_k(s^{(i)})}. \tag{49}$$

The terms A_j and B_j correspond to those in (40) and (41) and their product in (49) is the same for all paths at time k.

The soft output Viterbi algorithm (SOVA) can be implemented in the register exchange mode [23] or in the trace back mode. It will now be described for the latter mode using the log-likelihood algebra.

As shown in Fig. 7, we wish to obtain the soft output for bit $\hat{u}_k$, which the VA decides after a delay δ. The VA proceeds in the usual way by calculating the metrics for the ith path using (48). For each state it selects the path with the larger metric $M_k(s^{(i)})$. At time $k+\delta$ the VA has selected the maximum-likelihood (ML) path with index i_δ and has discarded the other path with index i'_δ ending at this state. Along the ML path i_δ, which decides the bit $\hat{u}_k$, $\delta+1$ nonsurviving paths i'_l with indices $l = 0, \cdots, \delta$ have been discarded. Define the metric difference as

$$\Delta^l_k = M_{k+l}(s^{(i_l)}) - M_{k+l}(s^{(i'_l)}) \ge 0. \tag{50}$$

Then the probability $P\,(\text{correct})$ that the path decision of the survivor was correct at time $k+l$ given $\boldsymbol{y}_{j\le k+l}$, is from (49)

$$\begin{aligned} P\,(\text{correct}) &= \frac{p\,(\text{path}\, i_l, \boldsymbol{y}_{j\le k+l})}{p\,(\text{path}\, i_l, \boldsymbol{y}_{j\le k+l}) + p\,(\text{path}\, i'_l, \boldsymbol{y}_{j\le k+l})} \\ &= \frac{\exp\,(M_{k+l}(s^{(i_l)}))}{\exp\,(M_{k+l}(s^{(i_l)})) + \exp\,(M_{k+l}(s^{(i'_l)}))} \\ &= \frac{\exp\,(\Delta^l_k)}{1+\exp\,(\Delta^l_k)}. \end{aligned} \tag{51}$$

Therefore, the likelihood ratio or "soft value" of this binary path decision is Δ^l_k, because

$$\log \frac{P\,(\text{correct})}{1 - P\,(\text{correct})} = \Delta^l_k. \tag{52}$$

Furthermore, it was shown in [12] that the soft output of the VA is the decision $\hat{u}_k$ times the L-value of the errors and can finally be approximated by

$$L(\hat{u}_k) \approx \hat{u}_k \sum_{l=0}^{\delta}\!\!\boxplus\, \Delta_k^l \approx \hat{u}_k \cdot \min_{l=0,\cdots,\delta} \Delta_k^l. \tag{53}$$

The sum and the minimum is only over those nonsurviving paths which would have led to a different decision $\hat{u}_k$. Thus we have the same hard decisions as the classical VA, and the reliability of the decisions is obtained by taking the minimum of the relevant metric differences along the ML path.

For a systematic convolutional code it can be seen from Fig. 7, using (48) and (50), that each of the Δ_k^l has the following structure:

$$\begin{aligned}\Delta_k^l = &\left(M_{j<k}^{(1)} - M_{j<k}^{(2)}\right) + \left(M_{k<j<k+l}^{(1)} - M_{k<j<k+l}^{(2)}\right)\\ &+ \frac{1}{2}\sum_{\nu=2}^{n} L_c y_{k,\nu}\left(x_{k,\nu}^{(1)} - x_{k,\nu}^{(2)}\right)\\ &+ \frac{1}{2} L_c y_{k,1}(\hat{u}_k - (-\hat{u}_k))\\ &+ \frac{1}{2} L(u_k)(\hat{u}_k - (-\hat{u}_k)).\end{aligned} \tag{54}$$

Therefore, the minimum value in (53) has the same structure. Thus the SOVA output in its approximate version in (53) has the format

$$L_{\text{SOVA}}(\hat{u}_k) = L_c y_{k,1} + L(u_k) + \underbrace{\hat{u}_k \cdot \{\text{first 3 terms in (47)}\}}_{L_e(\hat{u}_k)} \tag{55}$$

and preserves the desired additive structure of (20). Consequently, we subtract the input values from the soft output of the SOVA and obtain the extrinsic information to be used in the metrics of the succeeding decoder (see Fig. 4). In this case, the extrinsic term in (55) is weakly correlated to the other two terms. Furthermore, it has been shown that for small memories the SOVA is roughly half as complex as the Log-MAP algorithm [21].

C. MAP Decoding Rule for Linear Binary Block Codes

The results of Sections III-A and III-B can be applied to any code for which a trellis, especially a binary trellis, can be drawn. It is well known that the codewords of a linear binary (N, K) block code $\mathcal{C}$ can be represented as paths through a trellis of depth N with at most 2^{N-K} states [2], [9], [24]. For the construction of the trellis, the systematic $\boldsymbol{H}$-matrix of the code is used and this results in a trellis with an irregular structure as opposed to the regular trellis of the convolutional codes. Each transition between two states is labeled with the appropriate codeword symbol x_k, where the first K symbols are equal to the information bit u_k and the following $N-K$ symbols represent parity bits. Hence two paths leave each existing state during the information part, whereas in the parity part only one path leaves each present state. (For convolutional codes this is the case in the last m time instants of a terminated trellis.) The branch transition probability in (36) for systematic block codes with statistically independent information bits can therefore be written as

$$\gamma_k(s', s) = P(s|s') \cdot p(y_k|s', s) = p(x_k; y_k) \tag{56}$$

with $p(x_k; y_k)$ defined as

$$p(x_k; y_k) \triangleq \begin{cases} p(y_k|x_k) \cdot P(u_k), & 1 \le k \le K \\ p(y_k|x_k), & K+1 \le k \le N. \end{cases} \tag{57}$$

Note that the branch transition probability is only defined when there is a transition from state s' at time $k-1$ to state s at time k. The probability of $P(u_k)$ is calculated according to (40). For the calculation of the conditioned probability $p(y_k|x_k)$ we use a similar formula as in (41) but without the summation term. Furthermore, we obtain for the log-likelihood ratio associated with definition (57)

$$L(x_k; y_k) = \log\left(p(x_k = +1; y_k)/p(x_k = -1; y_k)\right)$$

$$L(x_k; y_k) \triangleq \begin{cases} L_c y_k + L(u_k), & 1 \le k \le K \\ L_c y_k, & K+1 \le k \le N. \end{cases} \tag{58}$$

Omitting the terms which are equal for all transitions from time $k-1$ to time k and using the preceding definition of $L(x_k; y_k)$, the branch transition operation to be used in (37) and (38) can be written as $\exp\left(L(x_k; y_k)x_k/2\right)$.

The forward recursion and the backward recursion of the "symbol-by-symbol" MAP algorithm are performed using (37) and (38). In analogy to (44), the MAP rule for block codes can be written as

$$L(\hat{u}_k) = L_c \cdot y_k + L(u_k) + \log \frac{\sum\limits_{\substack{(s',s)\\ u_k=+1}} \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum\limits_{\substack{(s',s)\\ u_k=-1}} \alpha_{k-1}(s') \cdot \beta_k(s)}. \tag{59}$$

Using the approximation described in Section III-A, the Log-MAP realization for systematic block codes results in

$$\begin{aligned}L_{\text{Log-MAP}}(\hat{u}_k) = &\ L_c \cdot y_k + L(u_k)\\ &+ \max_{\substack{(s',s)\\ u_k=+1}} \left(\log \alpha_{k-1}(s') + \log \beta_k(s)\right)\\ &- \max_{\substack{(s',s)\\ u_k=-1}} \left(\log \alpha_{k-1}(s') + \log \beta_k(s)\right).\end{aligned} \tag{60}$$

with

$$\log \alpha_k(s) = \max_{s'}\left(\log \alpha_{k-1}(s') + \frac{1}{2} L(x_k; y_k) \cdot x_k\right) \tag{61}$$

and

$$\log \beta_{k-1}(s') = \max_{s}\left(\log \beta_k(s) + \frac{1}{2} L(x_k; y_k) \cdot x_k\right). \tag{62}$$

In the following we will consider further ways of implementing the MAP decoding rule for linear block codes including *a priori* information in a unified presentation. This goes beyond what is known in the literature [2], [25] and is necessary for the iterative decoding technique as described in Section II-C. The first one implements the original code and is closely related to the "symbol-by-symbol" MAP algorithm described in this section. The second one uses the dual code and both algorithms lead to the same result. Further details are given in [26].

D. Straightforward Implementation of the "Symbol-by-Symbol" MAP Algorithm

Using the definitions (57) and (58) from the previous section, (19) can also be written as

$$\begin{aligned} L(\hat{u}_k) &= \log \frac{\sum_{\boldsymbol{x}\in\mathcal{C},u_k=+1} P(\boldsymbol{x}|\boldsymbol{y})}{\sum_{\boldsymbol{x}\in\mathcal{C},u_k=-1} P(\boldsymbol{x}|\boldsymbol{y})} \\ &= \log \frac{\sum_{\boldsymbol{x}\in\mathcal{C},u_k=+1} \left(\prod_{j=1}^{N} p(y_j|x_j)\cdot \prod_{j=1}^{K} P(u_j)\right)}{\sum_{\boldsymbol{x}\in\mathcal{C},u_k=-1} \left(\prod_{j=1}^{N} p(y_j|x_j)\cdot \prod_{j=1}^{K} P(u_j)\right)} \\ &= \log \frac{p(u_k=+1;y_k)\cdot \sum_{\boldsymbol{x}\in\mathcal{C},u_k=+1} \prod_{j=1,j\neq k}^{N} p(x_j;y_j)}{p(u_k=-1;y_k)\cdot \sum_{\boldsymbol{x}\in\mathcal{C},u_k=-1} \prod_{j=1,j\neq k}^{N} p(x_j;y_j)} \\ &= L(u_k) + L_c y_k \\ &\quad + \underbrace{\log \frac{\sum_{\boldsymbol{x}\in\mathcal{C},u_k=+1} \prod_{j=1,j\neq k}^{N} \exp\left(L(x_j;y_j)x_j/2\right)}{\sum_{\boldsymbol{x}\in\mathcal{C},u_k=-1} \prod_{j=1,j\neq k}^{N} \exp\left(L(x_j;y_j)x_j/2\right)}}_{L_e(\hat{u}_k)}. \end{aligned} \tag{63}$$

For the evaluation of (63) it is useful to separate the codewords into two groups; one with all codewords having a "+1" at the kth position, and the other with all codewords having a "-1" at the kth position. This separation can already be implemented into the trellis by minor changes in the construction principle [2]. The trellis is now built up by using all columns of the $\boldsymbol{H} = (\boldsymbol{h}_1, \cdots, \boldsymbol{h}_N)$ matrix excluding the kth one, and additionally by storing every path ending at time N at state $S_N = \boldsymbol{h}_k$. From now on we will call the two possible ending states $S_{\text{end1}} = 0$ and $S_{\text{end2}} = \boldsymbol{h}_k$. The time steps in the trellis will be named after the corresponding column of the $\boldsymbol{H}$ matrix, so that the kth time instant will no longer appear in the trellis. The paths ending in the zero state S_{end1} represent the codewords with a "+1" at the kth position and the paths ending in the state S_{end2} represent the codewords with a "-1" at the kth position.

Using the notation of Section III-C, the numerator of (63) is equal to the forward metric of the ending state $S_{\text{end1}} = 0$, $\alpha_N(S_{\text{end1}})$, and the denominator of (63) is equal to the forward metric of the second ending state $S_{\text{end2}} = \boldsymbol{h}_k$, $\alpha_N(S_{\text{end2}})$. Equation (63) can then be written as

$$L(\hat{u}_k) = L_c y_k + L(u_k) + \log \alpha_N(S_{\text{end1}}) - \log \alpha_N(S_{\text{end2}}). \tag{64}$$

For the calculation of α_N we can either use the exact formula in (37) or its approximation in (61).

In general, one has to construct K different trellises to obtain the soft output $L(\hat{u}_k)$ for all information bits. For the class of cyclic codes the trellises for the different information bits are obtained by simply shifting the indices. Fig. 8 shows a block diagram for the calculation of the soft outputs $L(u_k)$ for the $(7,4,3)$ Hamming code in systematic form.

Summing up the results of the last two sections we have presented two ways of implementing the "symbol-by-symbol" MAP rule for linear binary systematic block codes. In both realizations we have to build up the trellis (or the modified trellis) for the original code with at most 2^{N-K} states. Following the ideas of Bahl *et al.* [9], the soft output for all information bits is calculated with one forward and one backward recursion in the trellis. Following the ideas of Battail *et al.* [2], the soft output for all information bits is calculated with K forward recursions in the modified trellis. The metric computations and their approximations are the same in both algorithms.

E. Implementing the "Symbol-by-Symbol" MAP Decoder Using the Dual Code

Hartmann and Rudolph [25] and Battail, *et al.* [2] found a way to calculate the probabilities $P(u_k = \pm 1|\boldsymbol{y})$ using the codewords of the dual code C'. In coding systems where the dual code has fewer codewords than the original code, i.e., if $N-K < x < K$, this results in a reduction of the decoding complexity.

Both publications only present the formula for equally probable information bits. Here we will present the extended formula, where the *a priori* information is also involved.

For the derivation of the formula given in the Appendix we follow the idea of Hartmann and Rudolph [25] and finally obtain (see (65) at the bottom of this page). Here $\boldsymbol{x}'$ denotes the codewords of the dual code C', and we use the index $i = 1$ for the all-"+1" codeword.

For i.i.d. information bits u_k, i.e., $L(u_k) = 0$, a similar relation can be found in [2]. Hartmann and Rudolph [25] were only interested in a "symbol-by-symbol" MAP decoder with hard outputs, and they did not investigate either soft-output information or *a priori* information. Both are crucial for iterative decoding.

$$L(\hat{u}_k) = L_c y_k + L(u_k) + \underbrace{\log \frac{1 + \sum_{i=2}^{2^{N-K}} \prod_{j=1,j\neq k}^{N} \left(\tanh\left(L(x_j;y_j)/2\right)\right)^{(1-x'_{ij})/2}}{1 - \sum_{i=2}^{2^{N-K}} (-x'_{ik}) \prod_{j=1,j\neq k}^{N} \left(\tanh\left(L(x_j;y_j)/2\right)\right)^{(1-x'_{ij})/2}}}_{L_e(\hat{u}_k)}. \tag{65}$$

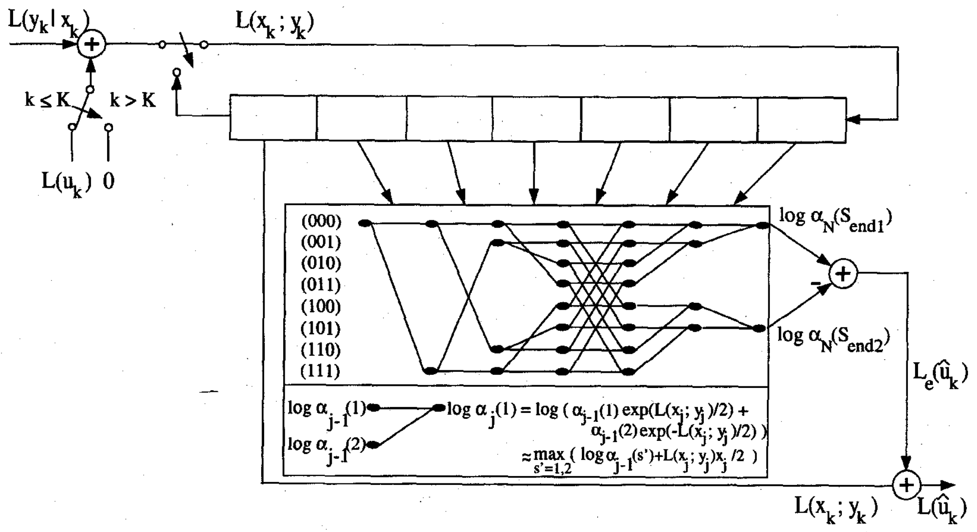

Fig. 8. 'Soft-in/soft-out' decoder for the $(7,4,3)$ Hamming code.

The dual code C' can also be represented in a trellis, now with a maximum of 2^K states. The corresponding metrics for every node are for the forward recursion

$$\tilde{\alpha}_k(s) = \sum_{s'} \tilde{\gamma}_k(s',s) \cdot \tilde{\alpha}_{k-1}(s') \tag{66}$$

and for the backward recursion

$$\tilde{\beta}_{k-1}(s') = \sum_{s} \tilde{\gamma}_k(s',s) \cdot \tilde{\beta}_k(s). \tag{67}$$

Whenever a transition between s' and s exists the branch transition operation is defined as

$$\tilde{\gamma}_k(s',s) = (\tanh(L(x_k;y_k)/2))^{(1-x'_k)/2}. \tag{68}$$

The forward and backward recursion is initialized with $\tilde{\alpha}_0(0) = 1$ and with $\tilde{\beta}_N(0) = 1$, respectively.

Again we have two ways of implementing the "symbol-by-symbol" MAP rule using the dual code. We can build up the full trellis for the dual codewords and run one forward and one backward recursion. Then the soft output for each information bit is calculated according to

$$\begin{aligned} L(\hat{u}_k) &= L_c y_k + L(u_k) \\ &\quad + \log \frac{\sum\limits_{(s',s)} \tilde{\alpha}_{k-1}(s')\tilde{\beta}_k(s)}{\sum\limits_{\substack{(s',s)\\ x'_k=+1}} \tilde{\alpha}_{k-1}(s')\tilde{\beta}_k(s) - \sum\limits_{\substack{(s',s)\\ x'_k=-1}} \tilde{\alpha}_{k-1}(s')\tilde{\beta}_k(s)}. \end{aligned} \tag{69}$$

The other way is to construct the modified trellis for the dual codewords according to Section III-D and to perform one forward recursion for each information bit. Then the soft output can be written as

$$\begin{aligned} L(\hat{u}_k) &= L_c y_k + L(u_k) + \log \frac{\tilde{\alpha}_N(S_{\text{end1}}) + \tilde{\alpha}_N(S_{\text{end2}})}{\tilde{\alpha}_N(S_{\text{end1}}) - \tilde{\alpha}_N(S_{\text{end2}})} \\ &= L_c y_k + L(u_k) + 2\,\text{artanh}\,(\tilde{\alpha}_N(S_{\text{end2}})/\tilde{\alpha}_N(S_{\text{end1}})). \end{aligned} \tag{70}$$

The dual code of a cyclic code is also cyclic, i.e., the modified trellises for every information symbol can be built one from the other by simply shifting the indices. In Fig. 9 we see a block diagram for the calculation of the soft outputs $L(x_k)$ with (70) for the $(7,4,3)$ Hamming code implementing its dual code, the $(7,3,4)$ maximum length code. The dual code implementation has more states than the straightforward implementation shown in Fig. 8. However, this is more than compensated by the fact that there are fewer code words to be checked, resulting in a very sparse trellis.

F. Simplifications of the "Symbol-by-Symbol" MAP Rule for Some Special Codes

The simple $(N, N-1)$ single parity check code (SPC) and the $(N,1)$ repetition code (RC) are both very weak codes, but they are excellent tools for constructing powerful concatenated codes, e.g., multidimensional product codes or Reed–Muller codes.

The dual code of the SPC has only two codewords, therefore the easiest way to obtain the "symbol-by-symbol" MAP rule is by using (65). With (11), this results in

$$L(\hat{u}_k) = L_c y_k + L(u_k) + \sum_{j=1, j\neq k}^{N}\boxplus\, L(x_j;y_j) \tag{71}$$

with the last term being the extrinsic information. Again we want to stress that one obtains the same formula using (63) for the MAP rule after some transformations. For the RC the implementation of the original code is the easiest way and we obtain as expected

$$L(\hat{u}_k) = \sum_{j=1}^{N} L(x_j;y_j). \tag{72}$$

For i.i.d. information bits only, both formulas can be found in the paper by Battail *et al.* [2], and also in the literature

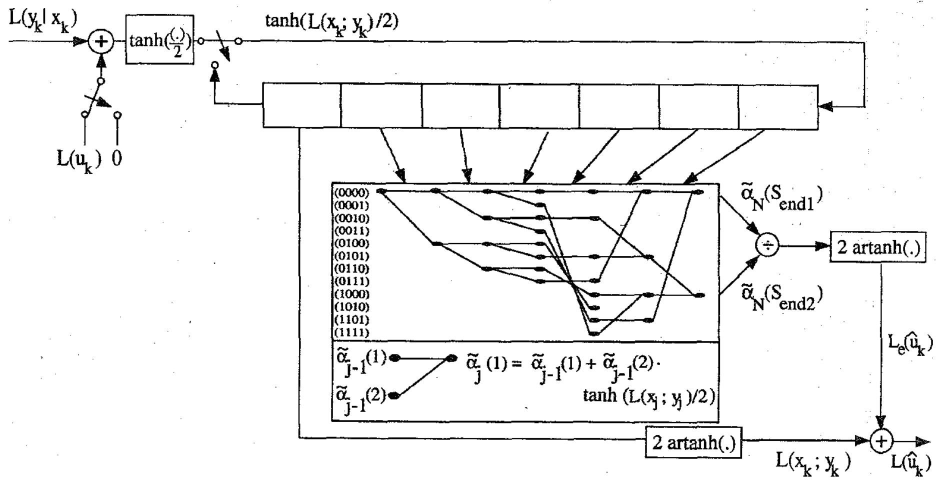

Fig. 9. "Soft-in/soft-out" decoder for the $(7,4,3)$ Hamming code using the dual code implementation.

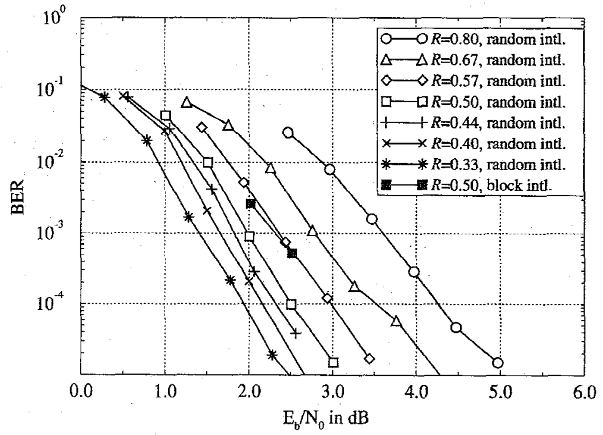

Fig. 10. Convolutional–convolutional component codes, component codes: memory 2, punctured from rate-$1/2$ mother code, interleaver size $K_1 \times K_2 = 30 \times 30$, six iterations with SOVA, random interleaver. For reference purposes the points □ at $R = 0.5$ are obtained with a block interleaver.

TABLE I
Parameter of the Convolutional Component Codes Used in the Simulations
(The rate of the mother code is $1/2$. The generator polynomials for the memory 2 codes are $g_0(D) = 1 + D + D^2$, $g_1(D) = 1 + D^2$, and for the memory 4 codes we used $g_0(D) = 1+D^3+D^4$, $g_1(D) = 1+D+D^2+D^4$)

rate	parity puncturing-pattern	total rate
8/9	10000000	0.80
8/10	10001000	0.67
8/11	10101000	0.57
8/12	10101010	0.50
8/13	11101010	0.44
8/14	11101110	0.40
8/16	11111111	0.33

about threshold decoding and its extension, the *a posteriori* probability (APP) decoder [27], [28].

Suboptimal solutions for the MAP decoder are possible by using only a limited number of codewords of the original code or the dual code in (63) or (65), respectively. A class of codes where the suboptimal solution is easily feasible and which is expected to give good results is the class of orthogonal codes. For a code with J orthogonal parity check equations, a suboptimal solution for the MAP rule can be implemented by using the approximate likelihood ratio derived from the J orthogonal parity check equations only

$$L(\hat{u}_k) \approx L(\hat{u}_k | J \text{ orth. parity checks})$$
$$\approx L_c y_k + L(u_k) + \sum_{j=1}^{J} \left(\sum_{i \in W_j}\!\boxplus\, L(x_i; y_i) \right). \quad (73)$$

W_j denotes the set of positions (without the kth one) of a "-1" in the jth orthogonal parity check equation.

IV. Simulation Results

A. Convolutional–Convolutional Component Codes

Fig. 10 shows the performance of convolutional–convolutional component codes, i.e., convolutional codes in both dimensions. Such codes were named "turbo" codes in [8], although there is nothing "turbo" in the code. Only the decoder uses feedback information and could be named "turbo" decoder in analogy with a turbo engine. The component code is realized by a feedback systematic encoder with punctured parity bits and its rate varies between $1/2$ and $8/9$.

The parameters of the code are given in Table I. Consequently, the overall code rate R is between $1/3$ and $8/10$. The block size $K_1 \times K_2$ of a random interleaver is fixed to be $30 \times 30 = 900$. This is a good compromise between the performance obtained by increasing the block size and reasonable delay.

For reasons of space, it is not possible to refer to the pseudo-random interleaver mapping in this paper. No special effort was made to optimize this pseudo-random interleaver as in [16]. For reference purpose we also show two points with the well-defined block interleaver in Fig. 10. Enlarging the block

TABLE II
AVERAGE NUMBER OF ITERATIONS USING THE STOP CRITERION IN (34)
(Overall code rate 1/2, component code rate 2/3, memory 2, interleaver size $K_1 \times K_2 = 30 \times 30$, SOVA, AWGN channel, more than 1000 bit error events per measured points.)

E_b/N_0	max. number of iterations: 6		max. number of iterations: 10	
	average number of iterations	relative increase of BER due to the use of the stop criterion	average number of iterations	relative increase of BER due to the use of the stop criterion
2.0	4.44		4.87	
2.5	3.42	$\leq 2\%$	3.51	$\leq 7\%$
3.0	2.73		2.74	

size from 200 to 680, 900, and 3200 improves the coding gain at a BER of 10^{-4} by 0.5, 0.7, and 1.2 dB, respectively. All the simulations were done for an AWGN channel. Each simulation point represents at least 10^3 bit error events. In [20] we presented simulation results for Rayleigh channels.

The convolutional code used in Fig. 10 has a memory $m = 2$. We also simulated a convolutional code with $m = 4$. However, at a BER of 10^{-4} the 16-state convolutional code behaves only slightly better (0.1 dB) than the 4-state convolutional code. The SOVA has a significantly lower complexity. Therefore, we used for the simulations the SOVA instead of the MAP algorithm. Furthermore, the loss of the SOVA compared to the MAP algorithm is only about 0.5 dB at a BER of 10^{-5} ($K_1 \times K_2 = 900$, $R = 1/2$).

The results with the convolutional component codes improve with the number of iterations. Most of the gain in iterative decoding is achieved by the first two or three iterations (one iteration includes a horizontal and the following vertical decoding). The results presented here are for six iterations. At a BER of 10^{-4} we achieve an additional gain of 2.2 dB by going from one iteration to six iterations. By ten more iterations the further improvement is only 0.2 dB.

In Fig. 10 we have presented results with a fixed number of iterations. If the cross-entropy is used as a stop criterion the number of iterations becomes a random variable resulting in a smaller average value.

This is shown in Table II. The usefulness of the stop criterion becomes clear for an operating point of 3 dB. Instead of ten iterations we only need an average of 2.74 iterations and the small increase in the error rate shows that we miss only a few errors.

B. Block–Block Component Codes

Fig. 11 shows the performance with simple Hamming codes as component codes. We used the dual code method in Section III-E for decoding. The resulting overall code rates R are in the range of 0.4 to 0.83. The same $(N, K, 3)$ Hamming code is used in both dimensions. The block size of the information part is $K \times K$. This means that every row or column consists of only one codeword and that the interleaver size varies. We performed simulations for block as well as for random interleaving. The results did not differ much. The curve in Fig. 11 shows the results after six iterations. If three iterations are used the loss is only about 0.2 dB at a BER of 10^{-4}. Note in Fig. 11 that for $R = 0.83$ the component code with

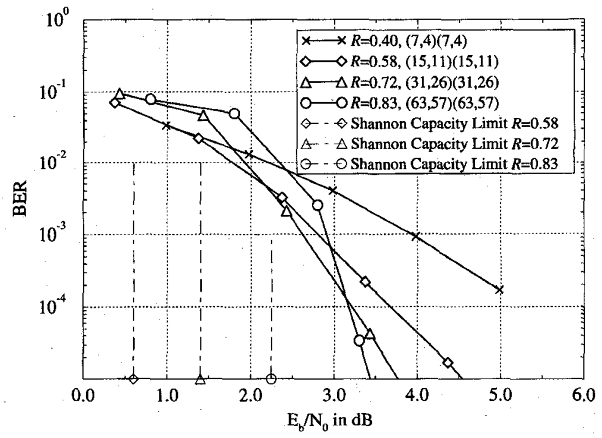

Fig. 11. Block–block component codes with overall code rate R, using Hamming codes as component codes, six iterations with (65), interleaver size $K \times K$.

the simple $(63, 57, 3)$ Hamming code as component code is only 1.2 dB away from the Shannon capacity limit at a BER of 10^{-5}, albeit with a block interleaver size of 3249 bits. The capacity limit is derived under the assumption of binary input/real output channel and for a capacity equal to the code rate.

The block–block decoding system with block interleaving fails in decoding a rectangular error pattern (with more than $d_{\min}/2$ errors in each direction, see Fig. 12(a). Therefore, we designed an improved interleaver similar to the one suggested by Nilsson [17]. The main idea indicated in Fig. 12(b) is to connect as many information bits as possible of a vertical codeword to other information bits of vertical codewords via the horizontal code. The improved interleaver consists of K_2 subblocks of size $K_1 \times l$, with $l \geq \max(K_1, K_2)$. The K_2 information bits of one vertical codeword are formed by the bits at the same position in each of the K_2 subblocks. After the encoding with the vertical code $\mathcal{C}^|$, the information bits in the subblocks are shifted column-wise. Enumerating the subblocks from 0 to $l - 1$ the interleaving can be described as follows: In the 0th subblock we have no permutation. In the first subblock we perform for every column (except the first one) a cyclic shift by one position relative to its left neighbor column. In the second subblock we perform for every column (except the first one) a cyclic shift by two positions relative to its left neighbor column, etc.. After these

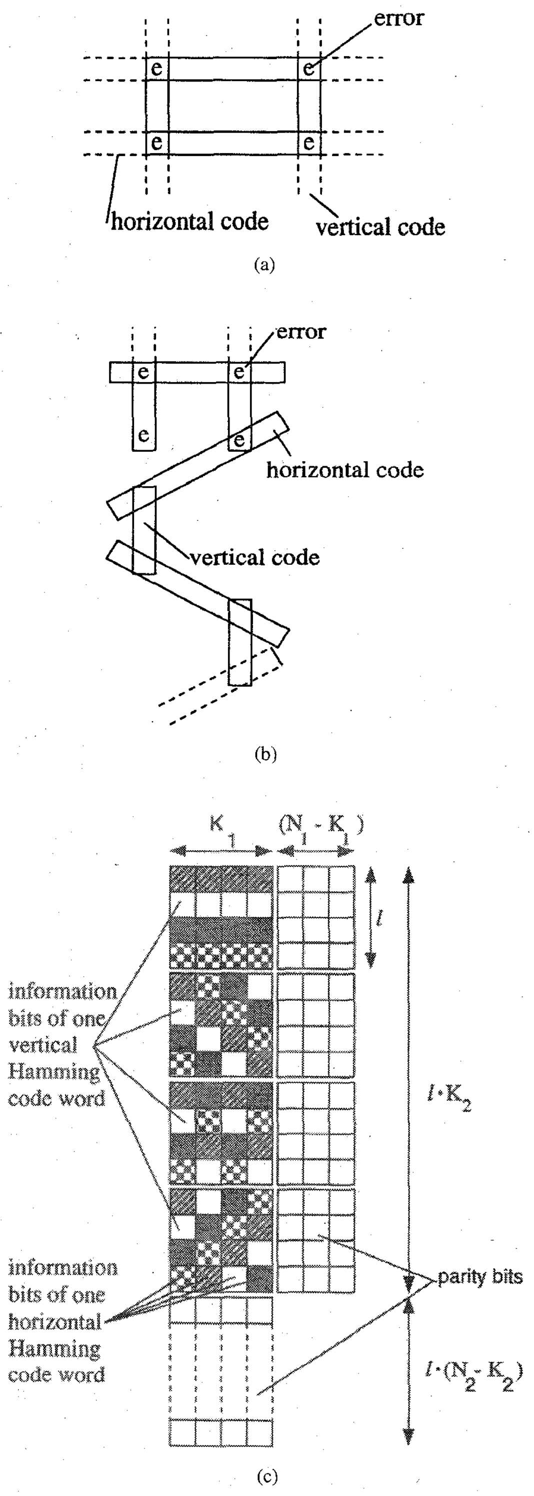

Fig. 12. (a) Noncorrectable error pattern for the block interleaver. (b) With the improved interleaver this error pattern is correctable. (c) Improved interleaving scheme for the $(7, 4, 3)$ Hamming code with $l = 4$.

permutations, the horizontal code is taken over each of the $l \cdot K_2$ rows of the subblocks. If we choose l to be the smallest prime number with $l \geq \max(K_1, K_2)$, then we can correct every rectangular error pattern with more than $d_{\min}/2$ errors in each direction. Otherwise, some of the rectangular error patterns might remain uncorrectable. Fig. 12(c) shows the whole improved interleaving scheme for the $(7, 4, 3)$ Hamming code with $l = 4$. The results given in Fig. 13 are obtained using the improved interleaver with $l = K$. We also simulated the improved interleaver with $l = 5$ for the $(7, 4, 3)$ Hamming code and with $l = 29$ for the $(31, 26, 3)$ Hamming code, but the results showed no significant difference compared to those presented in Fig. 13.

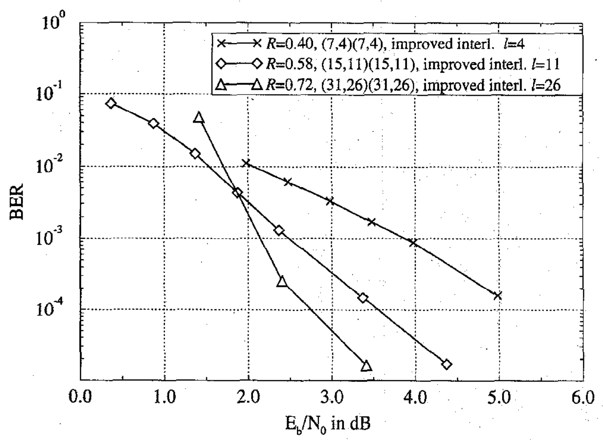

Fig. 13. Block–block component codes with rate R, using Hamming codes as component codes, six iterations with (65), improved interleaver of size $K \times (l \cdot K)$ as in Fig. 12.

After submission of the paper we became aware of the paper [29], in which a modified Chase algorithm is used for iterative decoding. However, in [29] they use 'factors optimized by simulation' for weighting the soft information. Although a full product code with a lower rate is used, their results are worse by 0.4 dB at a BER above 10^{-5}.

C. Discussion of the Simulation Results

The simulation results point out that convolutional–convolutional component codes perform well at low code rates. On the other hand, Hamming–Hamming component codes perform well at high rates. For a given BER it is possible to define a "threshold rate": for rates smaller than this value, one should use convolutional–convolutional codes, and for rates greater than this rate it is better to choose block–block codes. For the simulated code combinations the threshold rate is 0.67 at a BER of 10^{-4}. This is because high rate punctured convolutional codes are very weak as component codes, whereas good high-rate block codes exist which can be decoded with a reasonable complexity using the method given in Section III-E.

The combination of convolutional and Hamming codes is also possible and we performed several simulations. However, our simulations showed that this combination is worse than a convolutional–convolutional or Hamming–Hamming code. Only for rates near the threshold rate the combination of convolutional and Hamming codes performs as well as the Hamming–Hamming codes. In our simulations we used the suboptimal SOVA and the optimal MAP algorithm for decoding the convolutional and the block code, respectively.

V. Conclusions

We have investigated several aspects of systematic two- or more dimensional codes decoded by iterative "soft-in/soft-out" decoders. Iterative decoding is possible for convolutional codes in systematic feedback form, for any systematic block code or for combinations thereof using appropriate "soft-in/soft-out" algorithms derived from the MAP principle. The so-called "turbo" codes in [8] are just one example. For achieving nearly optimal performance the proper transferring of extrinsic information, from one iteration to the next, is crucial.

We conclude that very simple component codes such as the 4-state convolutional code and the Hamming codes are sufficient to achieve surprisingly good results. The minimum free distance of the component codes and the resulting minimum free distance of the two-dimensional code is not of prime importance for bit error rates above 10^{-5}. The interleaver, the soft-output component decoder, and the method of information transfer between the component decoders influences the performance. We urgently need tools to analyze and bound the performance in this range. As pointed out by Battail [4] very early, the asymptotic distance properties are of minor importance for this type of iteratively decoded codes. The variety of algorithms and combinations needs further investigation including the usage of other binary block codes and the search for suboptimal decoder algorithms.

However, with these simple codes and the associated decoding tools we achieve a BER of 10^{-4} with an E_b/N_0 of only 2 to 3 dB and rate 1/2. This is around the value determined by the cutoff rate and is only 2 dB away from the channel capacity limit. For higher rates and with simple block codes as component codes, we are even closer to the capacity limit.

Appendix
Derivation for the "Symbol-by-Symbol" MAP Decoding Rule Using the Dual Code

The main idea is to express $P(u_k = b|\boldsymbol{y}), b \in \mathrm{GF}(2)$ with the elements $\{+1, -1\}$, as a function over the whole code space and to use the finite Fourier transform. Hereby we follow the derivation found in [25]. Note that "+1" is the "null" element under the $\oplus$ addition, and that "-1" is the "one" element under the $\odot$ multiplication. For vectors $\boldsymbol{v} \odot \boldsymbol{w}$ denotes the scalar product in GF(2), where $\boldsymbol{v}$ and $\boldsymbol{w}$ are elements of the vector space V_N of all N-tuples over GF(2). We only consider linear binary block codes in systematic form, where the information bits u_k are statistically independent. The codeword $\boldsymbol{x}$ is transmitted over a time-discrete memoryless channel. Defining $\delta_{i,j} = 1$, if $i = j$ and $\delta_{i,j} = 0$, if $i \neq j$ and using

$$\boldsymbol{e}_k = (1 - 2\delta_{k,1}, \cdots, 1 - 2\delta_{k,N})$$

as the notation for the N-dimensional vector with "-1" in the kth postion and "+1" elsewhere we obtain

$$\begin{aligned} P(u_k = b|\boldsymbol{y}) &= \frac{1}{p(\boldsymbol{y})} \sum_{\substack{\boldsymbol{x} \in C \\ u_k = b}} p(\boldsymbol{x}, \boldsymbol{y}) \\ &= \frac{1}{p(\boldsymbol{y})} \sum_{\boldsymbol{x} \in C} p(\boldsymbol{x}, \boldsymbol{y}) \cdot \delta_{+1, (\boldsymbol{x} \odot \boldsymbol{e}_k \oplus b)}. \end{aligned} \tag{74}$$

Futhermore, we define the *a priori* probability $P_V(\boldsymbol{v})$ of any element $\boldsymbol{v}$ of the whole vector space V_N to be

$$P_V(\boldsymbol{v}) \triangleq \frac{1}{2^{N-K}} \prod_{j=1}^{K} P(v_j). \tag{75}$$

Hence, for the joint probability density function $p_{V,Y}(\boldsymbol{v}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{v}) \cdot P_V(\boldsymbol{v})$ using the definition of $p(v_j; y_j)$ in (57) and (75) we obtain

$$\begin{aligned} p_{V,Y}(\boldsymbol{v}, \boldsymbol{y}) &= \frac{1}{2^{N-K}} \prod_{j=1}^{N} p(y_j|v_j) \prod_{j=1}^{K} P(v_j) \\ &= 2^{K-N} \prod_{j=1}^{N} p(v_j; y_j) \end{aligned} \tag{76}$$

and for the joint probability function of codewords $p_{X,Y}(\boldsymbol{x}, \boldsymbol{y})$ we get

$$p_{X,Y}(\boldsymbol{x}, \boldsymbol{y}) = 2^{N-K} p_{V,Y}(\boldsymbol{v}, \boldsymbol{y})\big|_{\boldsymbol{v}=\boldsymbol{x}}. \tag{77}$$

Defining the finite Fourier transform

$$p_{V,Y}(\boldsymbol{v}, \boldsymbol{y}) = \frac{1}{2^N} \sum_{\boldsymbol{w} \in V_N} F(\boldsymbol{w}, \boldsymbol{y}) \boldsymbol{w} \odot \boldsymbol{v} \tag{78}$$

and using (77), $p_{X,Y}(\boldsymbol{x}, \boldsymbol{y})$ can be written as

$$p_{X,Y}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2^K} \sum_{\boldsymbol{w} \in V_N} F(\boldsymbol{w}, \boldsymbol{y}) \boldsymbol{w} \odot \boldsymbol{x} \tag{79}$$

where

$$\begin{aligned} F(\boldsymbol{w}, \boldsymbol{y}) &= \sum_{\boldsymbol{v} \in V_N} p_{V,Y}(\boldsymbol{v}, \boldsymbol{y}) \boldsymbol{w} \odot \boldsymbol{v} \\ &= 2^{K-N} \sum_{\boldsymbol{v} \in V_N} \prod_{j=1}^{N} p(v_j; y_j) w_j \odot v_j \\ &= 2^{K-N} \prod_{j=1}^{N} (p(+1; y_j) + p(-1; y_j) w_j). \end{aligned} \tag{80}$$

Furthermore, observe that

$$\delta_{+1, (\boldsymbol{x} \odot \boldsymbol{e}_k \oplus b)} = \frac{1}{2} \sum_{t \in \{+1, -1\}} t \odot (\boldsymbol{x} \odot \boldsymbol{e}_k \oplus b). \tag{81}$$

Using the relation of a dual code $\boldsymbol{x}' \in C'$

$$\sum_{\boldsymbol{x} \in C} \boldsymbol{w} \odot \boldsymbol{x} = \begin{cases} 2^K, & \boldsymbol{w} = \boldsymbol{x}' \in C' \\ 0, & \boldsymbol{w} \notin C' \end{cases} \tag{82}$$

$$L(\hat{u}_k) = \log \frac{P(u_k = +1|\boldsymbol{y})}{P(u_k = -1|\boldsymbol{y})} = \log \frac{\sum\limits_{t\in\{+1,-1\}} \sum\limits_{i=1}^{2^{N-K}} \prod\limits_{j=1}^{N} \left(1 + x'_{ij} \oplus t \odot (1 - 2\delta_{k,j}) \cdot e^{-L(x_j;y_j)}\right)}{\sum\limits_{t\in\{+1,-1\}} t \sum\limits_{i=1}^{2^{N-K}} \prod\limits_{j=1}^{N} \left(1 + x'_{ij} \oplus t \odot (1 - 2\delta_{k,j}) \cdot e^{-L(x_j;y_j)}\right)}. \tag{84}$$

and substituting (81) and (79) in (74) finally yields

$$P(u_k = b|\boldsymbol{y}) = \frac{1}{2p(\boldsymbol{y})} \sum_{t\in\{+1,-1\}} t \odot b \sum_{\boldsymbol{x}'\in C'} F(\boldsymbol{x}' \oplus t \odot \boldsymbol{e}_k, \boldsymbol{y}). \tag{83}$$

Applying (80) to (83) and using the definition of $L(x_j;y_j)$ in (58) we obtain (see (84) at the top of the page). After explicitly rewriting the first sum and extracting the kth term from the product, the numerator and the denominator in (84) can be written as

$$2 \cdot \sum_{i=1}^{2^{N-K}} \prod_{j=1,j\neq k}^{N} \left(1 + x'_{ij} \cdot e^{-L(x_j;y_j)}\right) \tag{85}$$

and

$$2 \cdot e^{-L(x_k;y_k)} \cdot \sum_{i=1}^{2^{N-K}} x'_{ik} \prod_{j=1,j\neq k}^{N} \left(1 + x'_{ij} \cdot e^{-L(x_j;y_j)}\right) \tag{86}$$

respectively. Dividing (85) and (86) by the term

$$\prod_{j=1,j\neq k}^{N} \left(1 + e^{-L(x_j;y_j)}\right)$$

and with the transformation

$$\prod_{j=1,j\neq k}^{N} \left(\frac{1 + x'_{ij} \cdot e^{-L(x_j;y_j)}}{1 + e^{-L(x_j;y_j)}}\right) = \prod_{j=1,j\neq k}^{N} \left(\frac{\exp(L(x_j;y_j)) - 1}{\exp(L(x_j;y_j)) + 1}\right)^{(1-x'_{ij})/2} \tag{87}$$

we finally obtain

$$L(\hat{u}_k) = L_c y_k + L(u_k) + \log \frac{\sum\limits_{i=1}^{2^{N-K}} \prod\limits_{j=1,j\neq k}^{N} \left(\frac{\exp(L(x_j;y_j))-1}{\exp(L(x_j;y_j))+1}\right)^{(1-x'_{ij})/2}}{\sum\limits_{i=1}^{2^{N-K}} x'_{ik} \prod\limits_{j=1,j\neq k}^{N} \left(\frac{\exp(L(x_j;y_j))-1}{\exp(L(x_j;y_j))+1}\right)^{(1-x'_{ij})/2}}. \tag{88}$$

Using again the relation

$$\tanh(x/2) = (e^x - 1)/(e^x + 1)$$

and using the index $i = 1$ for the all-"+1" codeword, the formula can also be written as

$$L(\hat{u}_k) = L_c y_k + L(u_k) + \log \frac{1 + \sum\limits_{i=2}^{2^{N-K}} \prod\limits_{j=1,j\neq k}^{N} (\tanh(L(x_j;y_j)/2))^{(1-x'_{ij})/2}}{1 - \sum\limits_{i=2}^{2^{N-K}} (-x'_{ik}) \prod\limits_{j=1,j\neq k}^{N} (\tanh(L(x_j;y_j)/2))^{(1-x'_{ij})/2}} \tag{89}$$

or using (13) as

$$L(\hat{u}_k) = L_c y_k + L(u_k) + \log \frac{1 + \sum\limits_{i=2}^{2^{N-K}} \tanh\left(\frac{1}{2} \sum\limits_{\substack{j=1,j\neq k,\\ x'_{ij}=-1}}^{N} \boxplus\, L(x_j;y_j)\right)}{1 - \sum\limits_{i=2}^{2^{N-K}} (-x'_{ik}) \tanh\left(\frac{1}{2} \sum\limits_{\substack{j=1,j\neq k,\\ x'_{ij}=-1}}^{N} \boxplus\, L(x_j;y_j)\right)} \tag{90}$$

which allows the approximation given in (12).

VII. Acknowledgement

The authors wish to thank Prof. J. B. Anderson and two anonymous reviewers whose comments were very helpful in improving an earlier version of this paper.

References

[1] J. Hagenauer, E. Offer, and L. Papke, "Matching Viterbi decoders and Reed–Solomon decoders in a concatenated system," in *Reed–Solomon Codes and Their Applications*, S. Wicker and V. K. Bhargava, Eds. New York: IEEE Press, 1994, ch. 11, pp. 242–271.

[2] G. Battail, M. C. Decouvelaere, and P. Godlewski, "Replication decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 332–345, May 1979.

[3] G. Battail and H. M. S. El-Sherbini, "Coding for radio channels," *Ann. Télécommun.*, vol. 37, nos. 1-2, pp. 75–96, Jan./Feb. 1982.

[4] G. Battail, "Building long codes by combination of simple ones, thanks to weighted-output decoding," in *Proc., URSI ISSSE* (Erlangen, Germany, Sept. 1989), pp. 634–637.

[5] J. Lodge, P. Hoeher, and J. Hagenauer, "The decoding of multidimensional codes using separable MAP 'filters'," in *Proc. 16th Biennial Symp. on Communications* (Queen's University, Kingston, Ont., Canada, May 1992), pp. 343–346.

[6] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer, "Separable MAP 'filters' for the decoding of product and concatenated codes," in *Proc., IEEE Int. Conf. on Communications* (Geneva, Switzerland, May 1993), pp. 1740–1745.

[7] J. Hagenauer and P. Hoeher, "Concatenated Viterbi-decoding," in *Proc. 4. Joint Swedish–Soviet Int. Workshop on Information Theory* (Gotland, Sweden, Aug. 1989), pp. 29–33.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes(1)," in *Proc., IEEE Int. Conf. on Communications* (Geneva, Switzerland, May 1993), pp. 1064–1070.

[9] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[10] S. Shamai (Shitz) and S. Verdú, "Capacity of channels with uncoded side information,"*European Trans. Telecommun. (ETT)* (Special Issue on Turbo Decoding), Sept. 1995.

[11] J. Hagenauer and L. Papke, "Decoding 'Turbo' codes with the soft-output Viterbi algorithm (SOVA)," in *Proc. Int. Symp. on Information Theory* (Trondheim, Norway, June 1994), p. 164.

[12] J. Hagenauer, "Source-controlled channel decoding," *IEEE Trans. Commun.*, vol. 43, no. 9, pp. 2449–2457, Sept. 1995.

[13] G. Battail and R. Sfez, "Suboptimum decoding using Kullback principle," in *Lecture Notes in Computer Science, no. 313*, B. Bouchon *et al.*, Eds. Berlin: Springer-Verlag, 1988, pp. 93–101.

[14] M. Moher, "Decoding via cross-entropy minimization," in *Proc., IEEE Globecom Conf.* (Houston, TX, Dec. 1993), pp. 809–813.

[15] P. Guinand and J. Lodge, "Trellis termination for turbo codes," in *Proc., 17th Biennial Symp. on Communications* (Queen's University, Kingston, Ont., Canada, May 1994), pp. 389–392.

[16] P. Robertson, "Illuminating the structure of decoders for parallel concatenated recursive systematic (turbo) codes," in *Proc., IEEE Globecom Conf.* (San Francisco, CA, Dec. 1994), pp. 1298–1303.

[17] J. Nilsson and R. Kötter, "Iterative decoding of product code constructions," in *Proc., Int. Symp. on Information Theory and Its Applications* (Sydney, Australia, Nov. 1994), pp. 1059–1064.

[18] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," in *Abstracts of Papers, Int. Symp. Inform. Theory* (Asilomar, CA, Jan. 1972), p. 90.

[19] P. L. McAdam, L. Welch, and C. Weber, "M.A.P. bit decoding of convolutional codes," in *Abstracts of Papers, Int. Symp. on Information Theory* (Asilomar, CA, Jan. 1972), p. 91.

[20] J. Hagenauer, L. Papke, and P. Robertson, "Iterative ('turbo') decoding of systematic convolutional codes with the MAP and the SOVA algorithms," in *Proc., ITG-Fachtagung 'Codierung'* (Munich, Germany, Oct. 1994), pp. 21–29.

[21] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE Int. Conf. on Communications* (Seattle, WA, June 1995), pp. 1009–1013.

[22] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.

[23] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc., IEEE Globecom Conf.* (Dallas, TX, Nov. 1989), pp. 1680–1686.

[24] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 76–80, Jan. 1978.

[25] C. R. Hartmann and L. D. Rudolph, "An optimum symbol-by-symbol decoding rule for linear codes," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 514–517, Sept. 1976.

[26] E. Offer, "Decodierung mit Qualitätsinformation bei verketteten Codiersystemen," Ph.D. dissertation, submitted to Tech. Univ. Munich, Munich, Germany, July 1995.

[27] J. L. Massey, *Threshold Decoding*. Cambridge, MA: M.I.T. Press, 1963.

[28] G. C. Clark, Jr. and J. B. Cain, *Error-Correction Coding for Digital Communications*. New York: Plenum, 1982.

[29] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes," in *Proc., IEEE Globecom Conf.* (San Francisco, CA, Nov. 1994), pp. 339–343.