Chapter 10

Maximum-Likelihood Viterbi Decoding and Performance of Convolutional Codes

Für convolutional codes gibt es verschiedene Decodierprinzipien:

Maximum-Likelihood-Decodierung: Eine sehr elegante und zugleich aufwandsgünstige Realisierung der ML-Decodierung ermöglicht der Viterbi-Algorithmus, der seit 1967 bekannt ist. Teilweise wird auch der Begriff Viterbi-Decodierung verwendet. Dieses Verfahren ist von großer praktischer Bedeutung und kann neben der Decodierung von convolutional codes und Trelliscodes prinzipiell bei allen Systemen angewendet werden, die eine Trellisstruktur aufweisen. Dazu zählen auch Interferenz-Verzerrungen, digitale modulation scheme mit Gedächtnis und sogar block codes, wie sich in den Abschnitten 11.4 bis 11.7 noch zeigen wird.

Die Viterbi-Decodierung ist bei hohen Übertragungsraten heute technisch realisierbar bis zu einer Gedächtnislänge von etwa $m = 6$ oder $m = 8$. Jede Erhöhung von m um 1 verdoppelt den Aufwand. Der Viterbi-Algorithmus verarbeitet fast ohne Mehraufwand Soft-Decision als Input mit entsprechenden coding gains und kann andererseits Soft-Decision als Output erzeugen.

Sequentielle Decodierung: Hierbei sind wesentlich größere Gedächtnislängen bis etwa $m = 100$ möglich. Der Rechenaufwand für die Decodierung ist nicht konstant wie beim Viterbi-Algorithmus, sondern variiert mit der Anzahl der Fehler. Der Rechenaufwand kann auch von außen vorgegeben werden und erlaubt damit eine Steuerung des coding gains. Die sequentielle Decodierung geht zurück auf Wozencraft (1957) und wird üblicherweise mit dem Fano-Algorithmus (1963) oder dem Stack-Algorithmus (Zigangirov 1966, Jelinek 1969) realisiert.

Algebraische Decodierung: Damit wird eine Klasse von einfachen und suboptimalen Decodierverfahren bezeichnet, die heute nur noch in Sonderfällen

verwendet werden, wo es auf allereinfachste Realisierung oder extrem hohe Übertragungsraten ankommt. Bekannt wurden diese Verfahren durch Massey (1963).

Die Schwerpunkte dieses Kapitels bilden die ML-Decodierung mit dem Viterbi-Algorithmus einschließlich der Implementierungsaspekte und der Erzeugung von Soft-Decision-Output sowie die Berechnung der post-decoding error probability einschließlich der entstehenden error structures. Daneben wird das Konzept of concatenated coding eingeführt und schließlich erfolgt ein Vergleich zwischen block codes and convolutional codes.

10.1 Maximum-Likelihood-Decoding and the Viterbi Metric

Für die Herleitung der ML-Decodierung kann gedanklich ein truncated convolutional code vorausgesetzt werden, der dann ein spezieller block code ist. Nach Theorem 1.2 wird zur Empfangsfolge $\boldsymbol{y}$ als ML-Schätzung $\hat{\boldsymbol{a}}$ diejenige Codefolge gewählt, bei der die transition probability maximal wird. Für alle Codefolgen $\boldsymbol{b}$ muß also gelten:

$$P_{y|x}(\boldsymbol{y}|\hat{\boldsymbol{a}}) \geq P_{y|x}(\boldsymbol{y}|\boldsymbol{b}). \tag{10.1.1}$$

Für den DMC zerfällt (factors) die transition probability for sequences nach Definition 1.1 in ein Produkt von einzelnen symbol transition probabilities.

$$P_{y|x}(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=0}^{l} P_{y|x}(y_i|x_i). \tag{10.1.2}$$

Dabei ist $l = (L+m)n - 1$, wenn nach L information bits terminiert wird, und i enumerates the sequences of received symbols and encoded bits. Anstelle von $P_{y|x}(\boldsymbol{y}|\boldsymbol{x})$ kann auch der skalierte Logarithmus maximiert werden ($\alpha > 0$, Rechnung in den reellen Zahlen):

$$\begin{aligned} \mu(\boldsymbol{y}|\boldsymbol{x}) &= \alpha \cdot \log P_{y|x}(\boldsymbol{y}|\boldsymbol{x}) + \beta & (10.1.3)\\ &= \alpha \cdot \sum_{i=0}^{l} \log P_{y|x}(y_i|x_i) + \beta \\ &= \sum_{i=0}^{l} \Big(\alpha \cdot \log P_{y|x}(y_i|x_i) + \beta_i \Big) \\ &= \sum_{i=0}^{l} \mu(y_i|x_i). & (10.1.4) \end{aligned}$$

Die Summe $\mu(\boldsymbol{y}|\boldsymbol{x})$ wird als *Viterbi-Metrik* des DMC bezeichnet, die natürlich vom verwendeten Code unabhängig ist. Die Summanden $\mu(y_i|x_i)$ werden als

Metrik-Inkremente bezeichnet. Die ML-Decodierung ist äquivalent mit der Maximierung der Viterbi-Metrik durch Wahl einer geeigneten Codefolge. Mit (9.1.4) wird eine *Inkrementalisierung* der Viterbi-Metrik gegeben. Die Größen $\alpha > 0$ und β_i sind dabei frei wählbar. Nur β_i darf von i bzw. y_i abhängen, aber natürlich nicht von x_i.

Für den BSC und den AWGN wird jetzt gezeigt, daß durch geeignete Wahl der Konstanten das Metrik-Inkrement in die Form $\mu(y_i|x_i) = x_i y_i$ gebracht werden kann:

Example 10.1. Für den BSC wird $\mathcal{A}_{\text{in}} = \mathcal{A}_{\text{out}} = \{+1, -1\}$ gesetzt. Für die Übergangswahrscheinlichkeit gilt nach Definition 1.2:

$$P(y|x) = \left\{ \begin{array}{ll} 1-p_e & x = y \\ p_e & x \neq y \end{array} \right\} = \left\{ \begin{array}{ll} 1-p_e & xy = +1 \\ p_e & xy = -1 \end{array} \right\}.$$

Mit $\alpha = 2/\log((1-p_e)/p_e) > 0$ (bei $p_e < 1/2$) und $\beta = -1 - \alpha \log p_e$ nimmt das Metrik-Inkrement folgende Form an:

$$\begin{aligned} \mu(y|x) &= \alpha \log P(y|x) + \beta \\ &= \left\{ \begin{array}{ll} \alpha \log(1-p_e) + \beta & xy = +1 \\ \alpha \log p_e + \beta & xy = -1 \end{array} \right\} \\ &= \left\{ \begin{array}{ll} \alpha \log((1-p_e)/p_e) - 1 & xy = +1 \\ -1 & xy = -1 \end{array} \right\} \\ &= \left\{ \begin{array}{ll} +1 & xy = +1 \\ -1 & xy = -1 \end{array} \right\} = xy. \end{aligned}$$

Wegen

$$d_H(x,y) = \left\{ \begin{array}{ll} 0 & xy = +1 \\ 1 & xy = -1 \end{array} \right\} = \frac{1-xy}{2}$$

ist die Maximierung der Viterbi-Metrik äquivalent mit der Minimierung des Hammingabstandes, wie es in Theorem 1.3 für die ML-Decodierung abgeleitet wurde. In den folgenden Beispielen wird allerdings $\mathcal{A}_{\text{in}} = \mathcal{A}_{\text{out}} = \{0, 1\}$ gesetzt und dann ist $\mu(y|x) = 1 - d_H(x,y)$ zu maximieren. Somit entspricht die Viterbi-Metrik der Anzahl der Übereinstimmungen zwischen der Empfangsfolge und der Codefolge zu einem Weg durch den Trellis.

In Abschnitt 9.7 wird noch eine Erweiterung zu einem zeitvarianten BSC betrachtet, bei dem die Bit-Fehlerwahrscheinlichkeit p_e von Kanalbenutzung zu Kanalbenutzung variieren kann. ■

Example 10.2. Für den AWGN channel wird $\mathcal{A}_{\text{in}} = \{+1, -1\}$ und $\mathcal{A}_{\text{out}} = \mathbb{R}$ gesetzt. Für die bedingte Verteilungsdichte gilt nach Definition 1.3:

$$f(y|x) = \frac{1}{\sqrt{\pi N_0}} \exp\left(-\frac{(y-x)^2}{N_0} \right).$$

Für die Viterbi-Metrik

$$\mu(y|x) = \alpha \log f(y|x) + \beta = \alpha \left(\ln \frac{1}{\sqrt{\pi N_0}} - \frac{y^2}{N_0} - \frac{x^2}{N_0} + \frac{2}{N_0} xy \right) + \beta$$

wird $\alpha = N_0/2 > 0$ gewählt. Stets gilt $x^2 = 1$ und somit folgt

$$\mu(y|x) = \frac{N_0}{2} \left(\ln \frac{1}{\sqrt{\pi N_0}} - \frac{y^2}{N_0} - \frac{1}{N_0} \right) + \beta + xy = xy,$$

sofern β geeignet gewählt wird. Die Viterbi-Metrik $\mu(\boldsymbol{y}|\boldsymbol{x})$ entspricht also einem Skalarprodukt. Bereits in (1.6.11) wurde die ML-Regel als Maximierung dieses Skalarprodukts formuliert. ■

Zusammenfassung: Sowohl beim BSC wie beim AWGN mit $\mathcal{A}_{\text{in}} = \{+1, -1\}$ ist zur Empfangsfolge $\boldsymbol{y}$ die Codefolge $\hat{\boldsymbol{a}}$ so zu wählen, daß das Skalarprodukt maximal wird:

$$\mu(\boldsymbol{y}|\hat{\boldsymbol{a}}) = \sum_{i=0}^{l} y_i \hat{a}_i = \sum_{i=0}^{l} \mu(y_i|\hat{a}_i). \tag{10.1.5}$$

Dies entspricht der Minimierung des Hammingabstandes beim BSC (Theorem 1.3) bzw. der Minimierung des euklidischen Abstandes beim AWGN (Theorem 1.4).

10.2 The Viterbi Algorithm

Das Grundprinzip des Viterbi algorithm for maximum-likelihood decoding wird in der nächsten Subsection zunächst für truncated convolutional codes eingeführt. In Subsection 10.2.2 werden wir dann die Besonderheiten bei non-truncated codes präsentieren und in Subsection 10.2.3 werden schließlich implementation and synchronization issues behandelt.

10.2.1 Truncated Convolutional Codes

Alle Kanten werden mit dem Metrik-Inkrement für einen Block beschriftet, denn die Inkrementalisierung erfolgt jetzt zweckmäßigerweise mit den Codeblöcken anstelle der Codebits,

$$\mu(\boldsymbol{y}_r|\boldsymbol{a}_r) = \sum_{\nu=1}^{n} y_{r,\nu} a_{r,\nu}. \tag{10.2.1}$$

Mit der Terminologie aus Abschnitt 8.6 ist die ML-Decodierung nun damit äquivalent, daß derjenige Weg durch den Trellis vom Anfangszustand $z_0 = \zeta_1$ bis zum Endzustand $z_{L+m} = \zeta_1$ gefunden wird, dessen Metrik bzw. dessen Summe der Inkremente maximal ist. Mit diesem Weg maximaler Metrik ist neben der geschätzten Codefolge auch zugleich die geschätzte Infofolge bekannt, so daß ein explizites Encoder-Inverses nicht erforderlich ist.

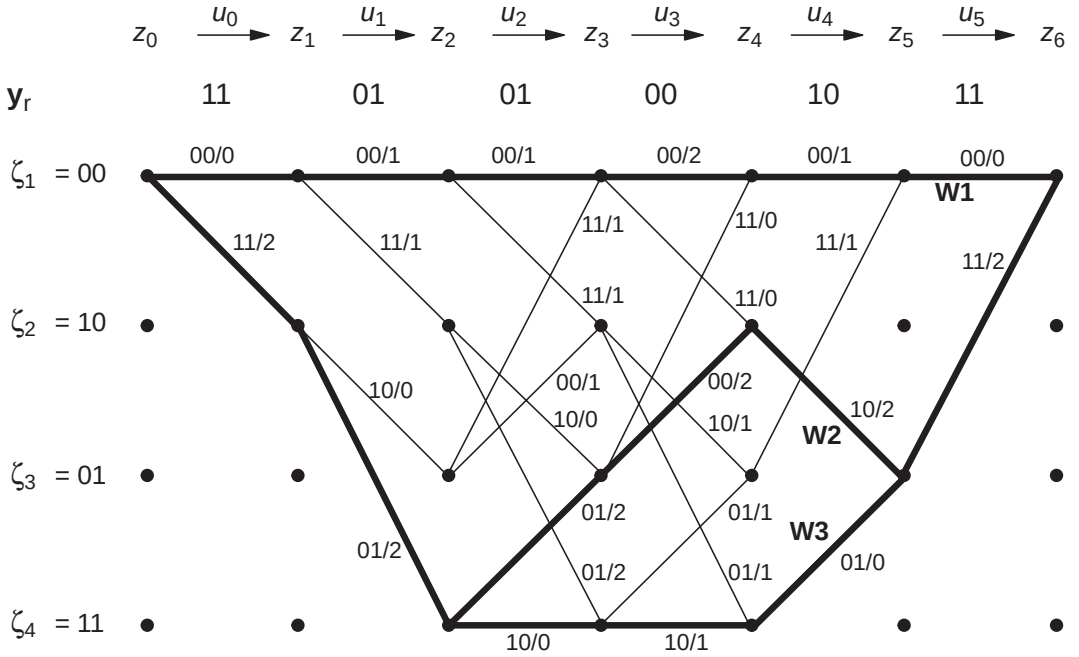

Figure 10.1. A trellis diagram for a correct transmission (standard example)

Example 10.3. In Bild 9.1 wird wie in Bild 8.9 als Empfangsfolge die ungestörte Codefolge 11, 01, 01, 00, 10, 11 zur Infofolge 1, 1, 0, 1, (0,0) angenommen. Die Kanten sind mit dem Codeblock sowie dem Metrik-Inkrement beschriftet. Dabei wird $\mathcal{A}_{\text{in}} = \mathcal{A}_{\text{out}} = \{0, 1\}$ gesetzt und $\mu(y|x) = 1 - d_H(x, y)$ ist zu maximieren. Für die drei dick markierten Wege gilt beispielsweise:

$$\mu(y|W_1) = 5 \quad , \quad \mu(y|W_2) = 12 \quad , \quad \mu(y|W_3) = 7.$$

W_2 entspricht der ungestört empfangenen Codefolge und hat maximales Gewicht. ■

Mit jeder Verlängerung des Trellis um ein Segment bzw. Codeblock bzw. information bit verdoppelt sich die Anzahl der zu untersuchenden Wege. Der Decodieraufwand wächst also exponentiell mit der Länge der information bit sequence. Der *Viterbi-Algorithmus* (VA) reduziert nun den Aufwand auf ein lineares Wachstum – und zwar ohne den geringsten Verlust an coding gain. Zudem kann auch bei nicht-terminierten Codes nach einer gewissen Verzögerung sukzessive entschieden werden, so daß dann der Rechenaufwand für die Decodierung zeitlich konstant bleibt.

Die Grundidee des Viterbi-Algorithmus ist sehr einfach: Es werden sukzessive Wege von $z_0 = \zeta_1$ zu allen Zuständen $z_r = \zeta_i$ betrachtet, wobei r von 1 bis $L + m$ läuft. Für jeden Weg ist eine Metrik

$$\mu_r(i) = \text{Metrik des Weges von } z_0 = \zeta_1 \text{ nach } z_r = \zeta_i \tag{10.2.2}$$

definiert. Beim Übergang von $z_r = \zeta_i$ nach $z_{r+1} = \zeta_j$ wird der Weg um ein Segment verlängert und die neue Metrik $\mu_{r+1}(j)$ ergibt sich aus der alten Metrik plus dem Inkrement. Von allen bei $z_r = \zeta_i$ ankommenden Wegen braucht nachfolgend nur noch der Weg mit maximaler Metrik berücksichtigt zu werden. Somit liegt bei jedem Zustand $z_r = \zeta_i$ nur ein überlebender Weg (*Survivor-Weg*) mit der Metrik $\mu_r(i)$ (*Survivor-Metrik*) an. Der neue Zustand $z_{r+1} = \zeta_j$ wird erreicht aus $z_r = \zeta_i$ und $z_r = \zeta_{i'}$ mit den Survivor-Metriken $\mu_r(i)$ und $\mu_r(i')$. Die neue Survivor-Metrik ergibt sich aus

$$\mu_{r+1}(j) = \max \left\{ \begin{array}{l} \mu_r(i) + \text{ Inkrement zur Kante } (\zeta_i, \zeta_j), \\ \mu_r(i') + \text{ Inkrement zur Kante } (\zeta_{i'}, \zeta_j) \end{array} \right\} \tag{10.2.3}$$

und der neue verlängerte Survivor-Weg ist somit der Weg mit der maximalen Metrik. Damit werden zu jedem Zeitpunkt r nur 2^m Wege der Länge r sowie insgesamt 2^m Survivor-Metriken gespeichert.

Obwohl dieses iterative Prinzip schon länger bekannt war, hat A.J.Viterbi 1967 als Erster dieses Prinzip auf die Decodierung von convolutional codes angewendet [227] und wurde damit zum Namensgeber des Algorithmus. Von G.D.Forney wurde 1973 die Äquivalenz des Viterbi-Algorithmus mit der ML-Decodierung nachgewiesen [170].

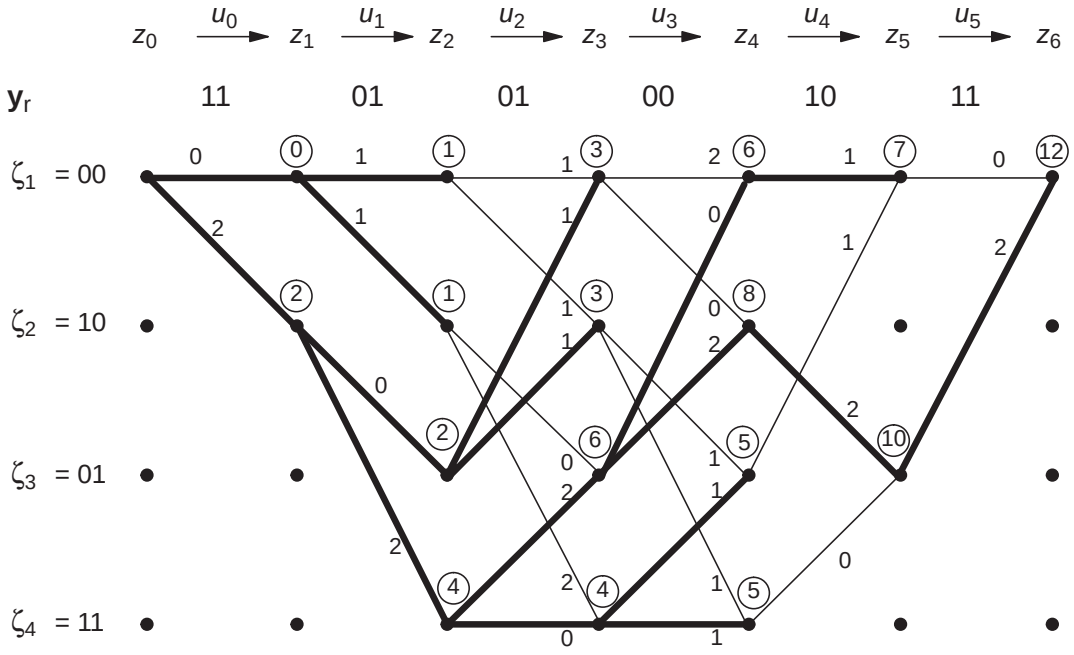

Figure 10.2. VA for a correct transmission (standard example with truncation)

Example 10.4. Der Viterbi-Algorithmus wird in Bild 9.2 für das Standardbeispiel demonstriert, wobei die gleiche ungestörte Empfangsfolge wie in Bild 8.9 und Bild 9.1 vorausgesetzt wird. Die Kanten sind mit den Inkrementen

beschriftet, die eingekreisten Zahlen sind die Survivor-Metriken und die dicken Kanten gehören zu den Survivor-Wegen. Im einzelnen gilt für die bei z_r anliegenden Survivor:

$$
\begin{aligned}
\text{Survivor bei } z_1 = \zeta_1 : &\quad \zeta_1, \zeta_1 \\
= \zeta_2 : &\quad \zeta_1, \zeta_2 \\
\text{Survivor bei } z_2 = \zeta_1 : &\quad \zeta_1, \zeta_1, \zeta_1 \\
= \zeta_2 : &\quad \zeta_1, \zeta_1, \zeta_2 \\
= \zeta_3 : &\quad \zeta_1, \zeta_2, \zeta_3 \\
= \zeta_4 : &\quad \zeta_1, \zeta_2, \zeta_4 \\
\text{Survivor bei } z_3 = \zeta_1 : &\quad \zeta_1, \zeta_2, \zeta_3, \zeta_1 \\
= \zeta_2 : &\quad \zeta_1, \zeta_2, \zeta_3, \zeta_2 \\
= \zeta_3 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_3 \\
= \zeta_4 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_4 \\
\text{Survivor bei } z_4 = \zeta_1 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_1 \\
= \zeta_2 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_2 \\
= \zeta_3 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_4, \zeta_3 \\
= \zeta_4 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_4, \zeta_4 \\
\text{Survivor bei } z_5 = \zeta_1 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_1, \zeta_1 \\
= \zeta_3 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_2, \zeta_3 \\
\text{Survivor bei } z_6 = \zeta_1 : &\quad \zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_2, \zeta_3, \zeta_1.
\end{aligned}
$$

Der bei $z_6 = \zeta_1$ verbleibende Survivor entspricht der ML-Schätzung. Da eine obere bzw. untere abgehende Kante dem information bit 0 bzw. 1 entspricht, ist die Infofolgen-Schätzung 1,1,0,1,(0,0) sofort klar. ■

Die für block codes sinnvolle Fragestellung "wieviel Fehler sind durch den ML-Decoder korrigierbar" ist bei convolutional codes wenig sinnvoll, denn um die Frage präzise zu stellen, muß sie so lauten: Sind alle Fehlermuster mit t Fehlern in einem Abschnitt der Länge h korrigierbar, sofern ein Abschnitt der Länge h' davor fehlerfrei war? Jedoch kann diese Frage entweder überhaupt nicht beantwortet werden oder es ergibt sich ein sehr kleiner Wert für t, wenn die Betonung auf *alle* Fehlermuster liegt. Obwohl die korrigierbaren Fehlermuster nicht analytisch erfaßbar sind, gelingt in Abschnitt 9.5 dennoch die exakte Berechnung der Fehlerwahrscheinlichkeit und des coding gains.

10.2.2 Non-Truncated Convolutional Codes

Ein umfangreiches Beispiel zum Viterbi-Algorithmus ohne Terminierung zeigt Bild 9.3, wobei wieder das Standardbeispiel vorausgesetzt wird. Als Zahlenbeispiel wird die Infofolge $u_0, \cdots, u_9 = 0111011000$ gewählt und in der Emp-

fangsfolge wird ein Fehler am Anfang angenommen:

$$\begin{aligned} \boldsymbol{a} &= 00\ 11\ 01\ 10\ 01\ 00\ 01\ 01\ 11\ 00 \\ \boldsymbol{y} &= 10\ 11\ 01\ 10\ 01\ 00\ 01\ 01\ 11\ 00. \end{aligned}$$

Ferner wird der Anfangszustand als unbekannt angenommen. Nach jedem Iterationsschritt werden alle Wegstücke gelöscht, die nicht zu den verbleibenden aktuellen Survivor-Wegen gehören. Bei Mehrdeutigkeiten sind jeweils alle möglichen Survivor angegeben. Am rechten Rand sind die Survivor-Metriken vermerkt. Bei z_9 verbleiben noch mehrere Survivor, aber alle bei z_{10} anliegenden Survivor laufen rückwärts betrachtet zu einem einzigen Weg zusammen, d.h. bei z_{10} kann eindeutig über $u_0, \ldots, u_7$ entschieden werden. Die Mehrdeutigkeit bei $z_1 = \zeta_1$ ist irrelevant, weil beide Kanten mit $u_0 = 0$ beschriftet sind. Wenn die Rückverfolgung des Survivors vom Zustand mit der maximalen Survivor-Metrik aus erfolgt, wird schon bei z_5 über u_0 richtig entschieden.

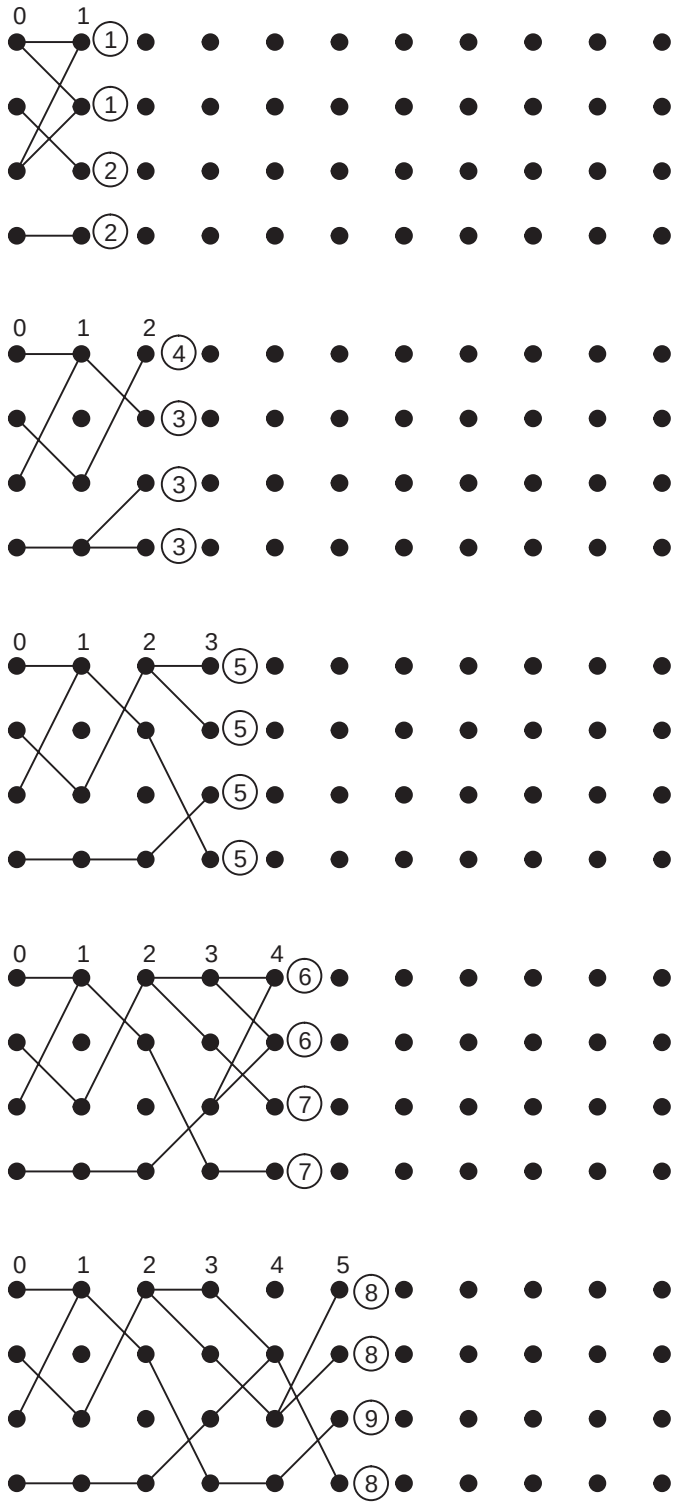

Figure 10.3a. VA for an erroneous transmission (standard example non-truncated)

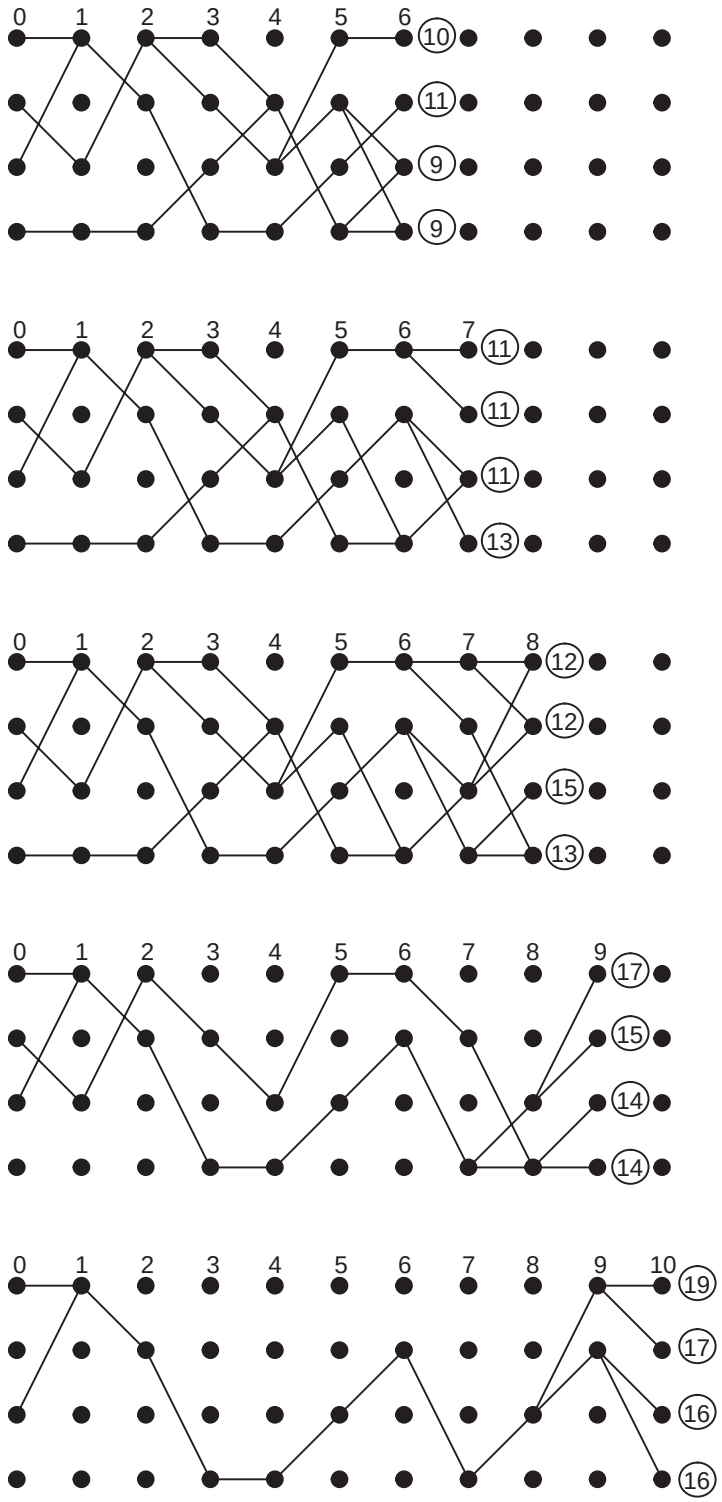

Figure 10.3b. Continuation of Figure 10.3a

Das Prinzip des Viterbi-Algorithmus für terminierte Codes wird mit diesem Beispiel deutlich: Unabhängig davon, von welchem Zustand $z_r = \zeta_i$ aus der Survivor zurückverfolgt wird, geht durch den Zustand z_{r-v} nur ein einziger Survivor, sofern v hinreichend groß gewählt wird. Also wird sich die Schätzung für das information bit u_{r-v} durch die Verlängerung der Survivor nach $z_{r+1}, z_{r+2}, \ldots$ nicht mehr verändern. Somit kann zum Zeitpunkt r eine endgültige Schätzung für u_{r-v} abgegeben werden. Diese Methode wird auch als *path memory truncation* bezeichnet.

Die Decodierverzögerung v wird *Rückgrifftiefe* (chainback memory length) genannt. Natürlich muß v so gewählt werden, daß mit hoher Wahrscheinlichkeit alle bei z_r anliegenden Survivor in z_{r-v} zusammenlaufen. Dann ergibt sich gegenüber einem unendlich langen Gedächtnis nur ein geringer Verlust im coding gain. Es besteht dann auch kein wesentlicher Unterschied zwischen den beiden folgenden Entscheidungsstrategien:

Bei der *Best State Rule* wird die maximale Metrik unter allen 2^m Survivor-Metriken gesucht und von dem entsprechenden Zustand aus wird der Survivor zurückverfolgt. Einfacher aber fast genauso gut ist die *Zero State Rule*, bei der die Rückverfolgung von einem fest eingestellten Zustand aus erfolgt. In den meisten Fällen wird durch die einfache Faustregel

$$v \approx 5 \cdot m \tag{10.2.4}$$

ein ausreichender Wert für v bei $R = 1/2$ gegeben. Bild 9.4 zeigt anhand von Simulationsergebnissen den Einfluß von v bei der Best State Rule und der Zero State Rule am Beispiel eines $R = 1/2$-Codes mit der Gedächtsnislänge $m = 4$ beim AWGN mit idealer Soft-Decision. Offensichtlich ist bei $v = 16$ die Zero State Rule noch deutlich schlechter als die Best State Rule, bei $v = 32$ sind dagegen beide Strategien nahezu identisch und durch Vergrößerung von v ergeben sich keine wesentlichen Verbesserungen (siehe auch Tabelle 9.1). Natürlich kann auch bei terminierten Codes schon nach v Trellissegmenten jeweils sukzessive entschieden werden, wenn es auf eine möglichst kurze Verzögerungszeit oder möglichst kleine Speicher ankommt.

Die Faustregel (9.3.1) ist für punctured convolutional codes allerdings nicht mehr anwendbar. One important aspect of puncturing is that the chainback memory length v must increase as the code rate increases. Whereas $v \approx 35 \ldots 40$ is adequate for rate-1/2 codes with $m = 6$, rate-3/4 decoders require $v \approx 70$ to be efficient, and rate-7/8 decoders should have $v \geq 90$. The Qualcomm implementation of the industry standard decoder with $m = 6$ and puncturing as given in Table 9.4 uses a chainback memory length of $v = 96$ [216]. Operation with higher code rates than 7/8 will result in a minor performance degradation when compared to the theoretical best.

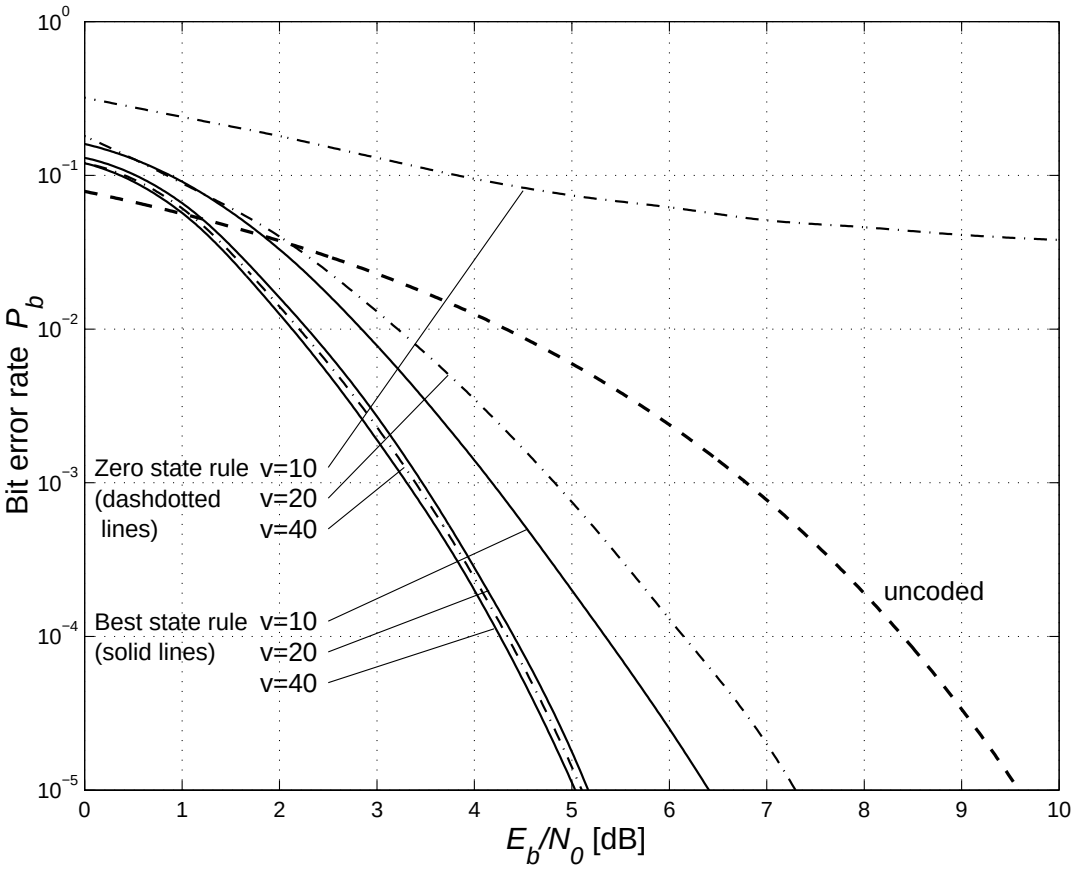

Figure 10.4. Comparison of VA traceback strategies ($R = 1/2, m = 4$)

10.2.3 Synchronisation and Implementation Considerations

In Bild 9.5 wird eine Aufgliederung des Viterbi-Decoders in einen Metrik-Prozessor und einen ACS-Prozessor (Add-Compare-Select) vorgeschlagen.

Im *Metrik-Prozessor* werden die Metrik-Inkremente berechnet. Normalerweise kann $n \leq m + 1$ vorausgesetzt werden und dann gibt es pro Segment zwar 2^{m+1} Kanten, aber nur 2^n verschiedene Inkremente, da es prinzipiell nur 2^n verschiedene Codeblöcke geben kann. Dabei wirkt sich der spezielle Code überhaupt nicht aus, da noch keine Zuordnung der Metrik-Inkremente zu den Kanten getroffen wird. Die tatsächliche Anzahl der Inkremente reduziert sich noch weiter, wenn Symmetrien ausgenutzt werden; beispielsweise gilt $\mu(y| - x) = -\mu(y|x)$ für die Form (9.1.5).

Für die Berechnung der Metrik-Inkremente kann bereits im Demodulator eine Quantisierung stattfinden. Nach Bild 2.2 erscheint eine oktale Quantisierung mit 3 Bit für den AWGN als ausreichend und tatsächlich wird dadurch die Fehlerwahrscheinlichkeit gegenüber idealer Soft-Decision nur gering verschlechtert (siehe auch Tabelle 9.1).

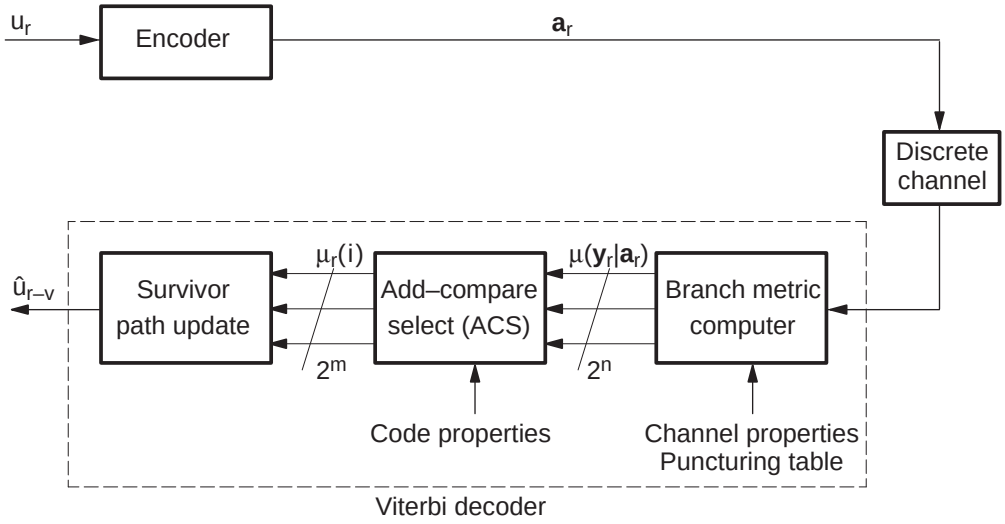

Figure 10.5. Block diagram of a practical Viterbi decoder

Bei punktierten Codes sollten die Alphabete mit $\mathcal{A}_{\text{in}} = \{+1, -1\}$ und $\mathcal{A}_{\text{out}} = \{+1, -1\}$ oder $\mathcal{A}_{\text{out}} = \mathbb{R}$ angenommen werden, weil dann die punktierten Stellen einfach durch $y_{r,\nu} = 0$ ergänzt werden können und somit in die Metrik-Inkremente $\mu(y|x) = xy$ nicht einfließen, d.h. die Punktierung ist damit äquivalent, daß an den punktierten Stellen Nullen als Empfangswerte eingefügt werden. Die Wechsel in der Coderate bei RCPC-Codes können also ohne wesentlichen Mehraufwand zu beliebigen Zeitpunkten erfolgen, über die allerdings der Metrik-Prozessor unterrichtet sein muß, um das Punktierungsschema umzuschalten.

Solange für den Kanal ein zeitinvarianter AWGN mit Hard-Decision oder Soft-Decision unterstellt werden kann, sind die Metrik-Inkremente unabhängig von den Kanaleigenschaften, also beispielsweise unabhängig von E_c/N_0. Zusammenfassung: Der Metrik-Prozessor wird nicht durch den Code, sondern allenfalls durch die Kanaleigenschaften und ein eventuelles Punktierungsschema geprägt.

Der *ACS-Prozessor* ist für die Berechnung der Survivor-Metriken und die Verlängerung der Survivor zuständig. Nur hier gehen die Codeeigenschaften ein, nämlich in der Zuordnung der 2^n Inkremente zu den 2^{m+1} Kanten.

Für die Organisation der Rechenoperationen und die Abspeicherung der Survivor sowie zur Rückverfolgung der Survivor bei sukzessiven Entscheidungen gibt es sehr unterschiedliche Lösungen. Dabei spielt es eine Rolle, ob Operationen parallel ausgeführt werden können. Ferner ist ein gewisser Austausch zwischen der Anzahl der arithmetischen Operationen und der Speichergröße möglich. Eine Übersicht dazu findet sich beispielsweise in [4, 25, 144].

Die Survivor-Metrik des richtigen Weges wächst gegenüber den Survivor-Metriken aller anderen Wege sehr stark an, so daß einer Überschreitung des Wertebereichs im ACS-Prozessor vorgebeugt werden muß. Die folgende Verschiebung aller Survivor-Metriken um einen konstanten Wert C zu beliebigen

Zeitpunkten hat auf die Decodierung keinen Einfluß:

$$\text{Ersetze } \mu_r(i) \text{ durch } \mu_r(i) - C \text{ für } i = 1, \ldots, 2^m. \tag{10.2.5}$$

Alle Survivor-Metriken, die eine untere Grenze $M_u < 0$ unterschreiten, werden auf diese untere Grenze zurückgesetzt, d.h. (9.4.1) wird verändert zu:

$$\text{Ersetze } \mu_r(i) \text{ durch } \max\{\mu_r(i) - C, M_u\} \text{ für } i = 1, \ldots, 2^m. \tag{10.2.6}$$

Bei einem sehr kleinen M_u wird hierdurch die Decodierung fast überhaupt nicht beeinflußt, aber die Survivor-Metriken befinden sich danach im Wertebereich zwischen M_u und 0, falls $C = \max_i \mu_r(i)$ gewählt wurde. Bei der Viterbi-Decodierung sind noch folgende Synchronisationsprobleme zu berücksichtigen:

Knotensynchronisation: Sofern nicht der Empfänger in eine laufende Übertragung eingeschaltet wird, tritt ein Anfangseffekt auf wie bereits mit Bild 8.9 erklärt wurde. Der Start beim Nullzustand kann im ACS-Prozessor automatisch dadurch berücksichtigt werden, daß die Survivor-Metriken zu Beginn mit

$$\mu_0(1) = 0 \quad , \quad \mu_0(2) = \cdots = \mu_0(2^m) = -\infty \tag{10.2.7}$$

(bzw. mit M_u statt $-\infty$) initialisiert werden. Falls das jedoch nicht möglich ist, entsteht nur ein kurzer Abschnitt mit hoher Fehlerrate, der nach ca. $5m$ Segmenten beendet ist und danach stellt sich wieder die normale niedrige Fehlerrate ein.

Symbolsynchronisation bzw. *Blocksynchronisation:* Der VA bzw. der Metrik-Prozessor muß die Unterteilung des Datenstroms in Blöcke der Länge n kennen, wobei $n = 2, 3, 4$ die typischen Werte sind. Eine falsche Synchronisation macht sich sehr schnell dadurch bemerkbar, daß es keinen dominierenden Weg, d.h. keine dominierende Survivor-Metrik gibt. Auch durch Re-Encodieren der geschätzten information bits und Vergleich mit den binär quantisierten Empfangsbits macht sich eine falsche Synchronisation durch eine Fehlerrate von rund 50% sofort bemerkbar. In diesem Fall wird die Unterteilung der Empfangswerte in Blöcke um eine Bitposition verschoben und und das vorangehend beschriebene Verfahren wird wiederholt. Auch eine Polaritätsvertauschung bei nicht-transparenten convolutional codes wird so entdeckt.

Rahmensynchronisation bei terminierten Codes: Dies wird bei den meisten Anwendungen durch Maßnahmen außerhalb der Kanalcodierung erreicht, beispielsweise durch uncoded synchronisation patterns. An einen derartigen Rahmen kann auch ein eventuelles Punktierungsschema angehängt werden.

Fazit: Alle Synchronisationsprobleme erweisen sich bei der Viterbi-Decodierung als harmlos und sind mit geringem Mehraufwand sicher zu beherrschen. Das ist ein ganz wesentlicher Vorteil gegenüber block codes.

In Tabelle 9.1 erfolgt eine Zusammenstellung zum Einfluß von Degradationen. Die Verluste durch Realisierungsvereinfachungen können bei gleicher Fehlerwahrscheinlichkeit durch eine entsprechende Erhöhung von E_b/N_0 ausgeglichen werden, wobei natürlich wieder der AWGN unterstellt wird. Die Werte in Tabelle 9.1 können als allgemeine Faustregeln für P_b im Bereich $10^{-3}, \ldots, 10^{-7}$ angesehen werden [25, 79, 114, 128].

Table 10.1. Degradations due to implementation losses

Realisierungsvereinfachung	Verlust
3-Bit-Quantisierung gegenüber idealer Soft-Decision	ca. 0.2 dB
1-Bit Hard-Decision gegenüber idealer Soft-Decision	ca. 2.2 dB
Rückgrifftiefe $v = 32$ gegenüber $v = \infty$ (bei $m = 6$)	ca. 0.2 dB

10.3 Calculation of Error Probabilities and Performance Results

In Subsection 10.3.1 wird zunächst die bit-error rate des Viterbi decoders analytisch berechnet, und zwar sowohl für allgemeine discrete channels und insbesondere für den binary AWGN channel einschließlich der asymptotic coding gains. Für praktische Anwendungen sehr wichtige performance results werden in Subsection 10.3.2 zusammengestellt, und in Subsection 10.3.3 wird die implementation and performance of Viterbi decoders für AWGN channels mit high-level modulation schemes betrachtet. Ein ebenfalls sehr wichtiger Gesichtspunkt sind die speziellen burst-error structures at the decoder output, die in Subsection 10.3.4 präsentiert werden.

10.3.1 Analytical Computation of the Bit-Error Rate

Die Berechnung der Fehlerwahrscheinlichkeit von convolutional codes erfolgt aus der Gewichtsfunktion, die alle dazu notwendigen Distanzeigenschaften enthält. Ähnlich wie bei den block codes können asymptotische coding gains für den AWGN bei Hard- und Soft-Decision angegeben werden.

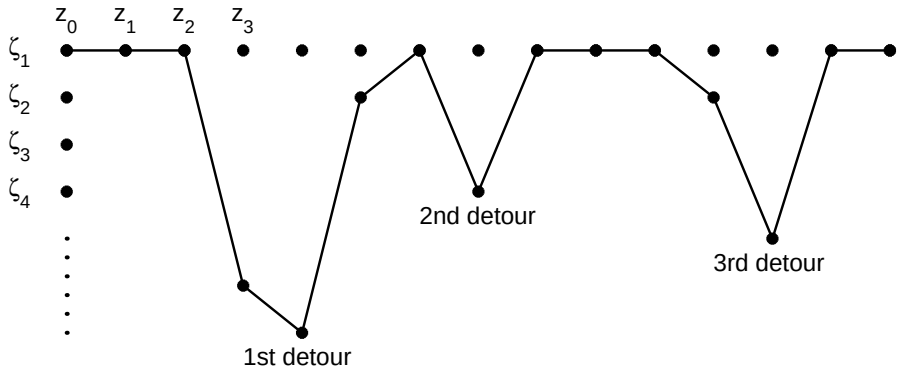

Figure 10.6. Error sequence as a series of independent detour paths

Es seien $\boldsymbol{a}_r$ die gesendeten und $\hat{\boldsymbol{a}}_r$ die geschätzten Codeblöcke bei ML-

Decodierung. Auch die Fehlerfolge

$$\boldsymbol{e}_r = \boldsymbol{a}_r - \hat{\boldsymbol{a}}_r \tag{10.3.1}$$

ist wegen der Linearität des Codes ebenfalls eine Codeblockfolge. Der Fehlerfolge entspricht umkehrbar eindeutig eine Zustandsfolge im Trellisdiagramm, wie es in Bild 9.6 dargestellt ist. Die Fehlerfolge kann interpretiert werden als eine durch Pausen unterbrochene Sequenz von Fundamentalwegen bzw. als Sequenz von *Fehlerereignissen*. Für die Berechnung der Fehlerwahrscheinlichkeit können diese Fehlerereignisse als statistisch unabhängig angenommen werden. Daraus ergibt sich der wesentliche Grundgedanke zur Herleitung der folgenden Ergebnisse, die den Sätzen 3.15 bis 3.17 für block codes entsprechen:

Theorem 10.1. *Vorausgesetzt wird ein $R = 1/n$-convolutional code mit der Gewichtsfunktion $T(D, I, J)$ und dem Distanzspektrum c_d gemäß Definition 8.4 sowie ML-Decodierung. Bei der Übertragung über den binären DMC mit der Bhattacharyya bound γ gemäß Definition 2.4 gilt für die information bit-Fehlerwahrscheinlichkeit P_b folgende Abschätzung:*

$$P_b \;\le\; \frac{\partial T}{\partial I}(\gamma, 1, 1) \;=\; \sum_{d=d_\mathrm{f}}^{\infty} \underbrace{\sum_{i,j} i \cdot t(d, i, j)}_{= c_d} \gamma^d. \tag{10.3.2}$$

Speziell für den AWGN mit idealer Soft-Decision gilt eine schärfere Abschätzung ($E_c = E_b/n$):

$$P_b \;\le\; \sum_{d=d_\mathrm{f}}^{\infty} c_d Q\left(\sqrt{2d\frac{E_c}{N_0}}\right). \tag{10.3.3}$$

Bei einem $R = k/n$-convolutional code sind die oberen Grenzen jeweils mit einem Faktor $1/k$ zu multiplizieren. Für die asymptotischen coding gains gilt (in dB):

$$G_{\mathrm{a,hard}} = 10 \cdot \log_{10}\left(\frac{Rd_\mathrm{f}}{2}\right) \quad , \quad G_{\mathrm{a,soft}} = 10 \cdot \log_{10}\Big(Rd_\mathrm{f}\Big). \tag{10.3.4}$$

Soft-Decision bringt gegenüber Hard-Decision einen Gewinn von 3 dB wie bei den block codes.

Proof. Mit dem Index r werden die Codeblöcke bzw. Segmente durchnumeriert. Mit dem Index l werden alle möglichen Fundamentalwege (FWeg) durchnumeriert (die zur Zeit Null beginnen). Der l-te FWeg hat das Codefolgengewicht d_l, das Infofolgengewicht i_l und die Länge j_l.

Es sei $P(r, l)$ die Wahrscheinlichkeit dafür, daß zur Zeit r der l-te FWeg beginnt. Anstelle des Nullweges wird also auf eine Codefolge der Länge $j_l n$ mit dem Hamminggewicht d_l entschieden. Diese Codefolge und der Nullweg bilden

einen $(j_l n, 1, d_l)_2$-block code mit zwei Codewörtern. Nach Theorem 3.16 bzw. nach Theorem 3.17 für den AWGN gilt dann:

$$P(r,l) \le \gamma^{d_l} \quad \text{bzw.} \quad P(r,l) \le Q\left(\sqrt{2d_l \frac{E_c}{N_0}}\right). \tag{10.3.5}$$

Zur Zählung der information bit-Fehler in der ML-Schätzung wird die Zufallsgröße

$$W_r = \left\{ \begin{array}{ll} i_l & \text{zur Zeit } r \text{ beginnt der } l\text{-te FWeg} \\ 0 & \text{zur Zeit } r \text{ beginnt kein FWeg} \end{array} \right\} \tag{10.3.6}$$

definiert, für deren Erwartungswert folgende Abschätzung gilt:

$$\begin{aligned} E(W_r) &= \sum_{l=0}^{\infty} i_l P(r,l) \le \sum_{l=0}^{\infty} i_l \gamma^{d_l} \\ &= \sum_{l=0}^{\infty} i \cdot \text{Anzahl(FWege mit } i = i_l \text{ und } d = d_l) \cdot \gamma^d \\ &= \sum_{d,i,j} i \cdot t(d,i,j) \gamma^d = \sum_{d=d_{\mathrm{f}}}^{\infty} c_d \gamma^d. \end{aligned}$$

Sei L eine große Zahl. Im Bereich $r = 0, 1, \ldots, L-1$ werden L information bits gesendet, von denen $W_0 + W_1 + \cdots + W_{L-1}$ falsch geschätzt werden – dabei wird vernachlässigt, daß hierbei auch einige Fehler mitgezählt werden, die erst ab $r = L$ auftreten. Also gilt:

$$P_b \;\approx\; \frac{W_0 + W_1 + \cdots + W_{L-1}}{L} \;\approx\; E(W_r) \;\le\; \sum_{d=d_{\mathrm{f}}}^{\infty} c_d \gamma^d. \tag{10.3.7}$$

Für $L \to \infty$ geht der Fehler bei den Approximationen gegen Null. Damit ist (9.5.2) bewiesen. Das Ergebnis (9.5.3) ergibt sich in gleicher Weise. Für Hard-Decision gilt asymptotisch nach (9.5.2):

$$\begin{aligned} P_b &\approx c_{d_{\mathrm{f}}} \gamma^{d_{\mathrm{f}}} \approx \gamma^{d_{\mathrm{f}}} = \left(\sqrt{4p_e(1-p_e)}\right)^{d_{\mathrm{f}}} \qquad \text{according to (2.3.5)} \\ &\approx p_e^{d_{\mathrm{f}}/2} = Q\left(\sqrt{2\frac{RE_b}{N_0}}\right)^{d_{\mathrm{f}}/2} \qquad \text{according to (1.3.12)} \\ &\approx e^{-Rd_{\mathrm{f}}/2 \cdot E_b/N_0} \qquad \text{according to (A.3.18).} \end{aligned}$$

Für Soft-Decision gilt asymptotisch nach (9.5.2) und (2.3.7):

$$P_b \approx c_{d_{\mathrm{f}}} \gamma^{d_{\mathrm{f}}} \approx \gamma^{d_{\mathrm{f}}} = \left(e^{-RE_b/N_0}\right)^{d_{\mathrm{f}}}.$$

Das gleiche Ergebnis folgt aus (9.5.3), d.h. die Abschätzungen (9.5.2) und (9.5.3) sind asymptotisch identisch. ■

Offensichtlich ist die Güte eines convolutional codes primär durch die freie Distanz d_{f} bestimmt. Allerdings sollte auch $c_{d_{\mathrm{f}}}$ klein sein. Je kleiner E_b/N_0 wird, desto mehr gewinnen auch die höheren Terme $c_{d_{\mathrm{f}}+1}, c_{d_{\mathrm{f}}+2}, \ldots$ an Bedeutung.

Die Approximation $P_b \approx \gamma^{d_{\mathrm{f}}}$ ist nur für den asymptotischen Fall $E_b/N_0 \to \infty$ zulässig. Denn mit wachsender Gedächtnislänge $m \to \infty$ gilt $d_{\mathrm{f}} \to \infty$ und somit $\gamma^{d_{\mathrm{f}}} \to 0$. Allein durch Erhöhung von m kann aber keine beliebig gute Übertragung erreicht werden, wie das Kanalcodierungstheorem aus Theorem 2.1 bei $R > C$ zeigt.

Example 10.5. Für das Standardbeispiel werden die Aussagen von Theorem 10.1 anhand von Figure 10.7 demonstriert. Vorweg bemerken wir daß die asymptotischen coding gains für dieses simple Beispiel mit $G_{\mathrm{a,hard}} = 0.97$ dB und $G_{\mathrm{a,soft}} = 3.98$ dB erstaunlich hohe Werte erreichen.

Die weight enumerator function ist nach Examples 9.10 and 9.11 in closed form bekannt und somit kann eine upper bound für die bit-error probability direkt berechnet werden ohne explizite Entwicklung of the distance spectrum. Nach Example 9.10 gilt:

$$\frac{\partial T}{\partial I}(D,1,1) = \frac{D^5}{1-4D(1-D)} = \frac{D^5}{(1-2D)^2}.$$

Für hard-decision decoding gilt $\gamma = \sqrt{4p_e(1-p_e)}$ und somit folgt aus Theorem 10.1

$$P_b \le \frac{\Big[4p_e(1-p_e)\Big]^{5/2}}{\Big[1-2\sqrt{4p_e(1-p_e)}\Big]^2}.$$

Für soft-decision decoding gilt $\gamma = e^{-RE_b/N_0} = e^{-E_b/2N_0}$ und somit folgt aus Theorem 10.1

$$P_b \le \frac{e^{-5E_b/2N_0}}{\Big[1-2e^{-E_b/2N_0}\Big]^2}.$$

In Figure 10.7 ist diese upper bound als Bhattacharyya bound bezeichnet (dash-dotted graph). Die exakte bit-error rate P_b (solid graph) wurde per computer simulation für den binary AWGN channel gewonnen, allerdings ist dann bei etwa einer bit-error rate von etwa 10^{-6} Schluss wenn man nicht extrem lange Simulationszeiten in Kauf nehmen will. Die Abweichung beträgt bei $P_b = 10^{-3}$ noch rund 1 dB, wird dann aber laufend kleiner und der asymptotische gap wird sogar zu Null.

Eine wesentlich schärfere bound ergibt sich mit (10.3.3). In Figure 10.5 wird die infinite Summation over d nach $d = d_{\mathrm{f}} = 5$, nach $d = 6$ und nach $d = 14$ abgebrochen. Für kleine bit-error rates reicht wirklich nur der Term $d = d_{\mathrm{f}}$, der jedoch bei größeren bit-error rates zu einer lower bound wird. Mit einigen

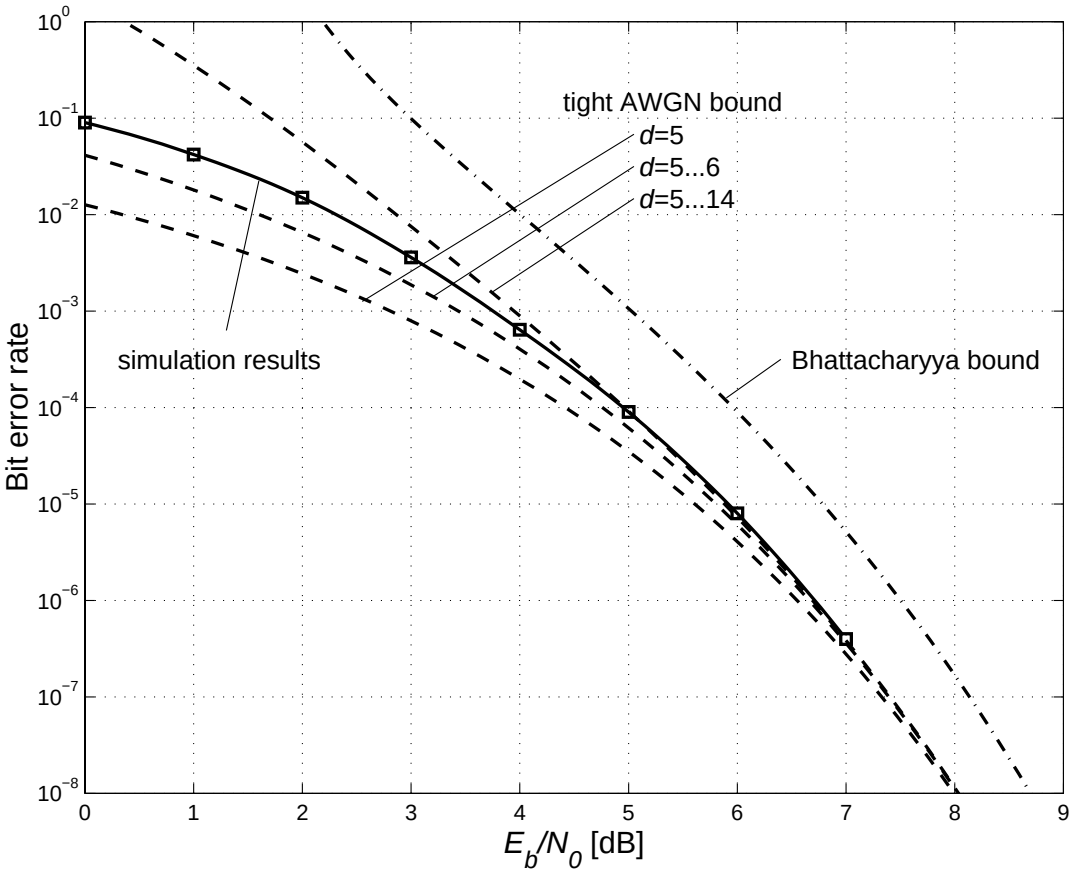

Figure 10.7. Exact results and bounds on the bit-error rate for the rate-1/2 standard example with $m = 2$ and soft-decision decoding

wenigen Termen wird schon unterhalb von 10^{-3} eine sehr genaue Approximation erreicht. Insgesamt ergänzen sich die Simulation im Bereich $P_b > 10^{-6}$ und die tight AWGN bound im Bereich $P_b < 10^{-4}$ perfekt. ■

10.3.2 Performance Results

Die Bilder 9.7 und 9.8 zeigen die Bit-Fehlerwahrscheinlichkeit P_b über E_b/N_0 der optimalen convolutional codes beim AWGN mit idealer Soft-Decision und ausreichend großer Rückgrifftiefe v. Wie in Example 10.5 schon erläutert wurde, sind die Kurven sind zusammengesetzt aus simulation results bei schlechten Kanälen (bei großem P_b kann die Fehlerrate durch die relative Häufigkeit geschätzt werden) und theoretischen Ergebnissen bei guten Kanälen (bei kleinem P_b brauchen vom distance spectrum nur die unteren Werte bekannt sein bzw. es kann durch den asymptotischen coding gain approximiert werden).

In Bild 9.7 werden convolutional codes verschiedener memory lengths bei konstanter code rate $R = 1/2$ miteinander verglichen. Die Erhöhung von m um 1 bedeutet zwar eine Aufwandsverdopplung im Decoder, aber auch einen coding

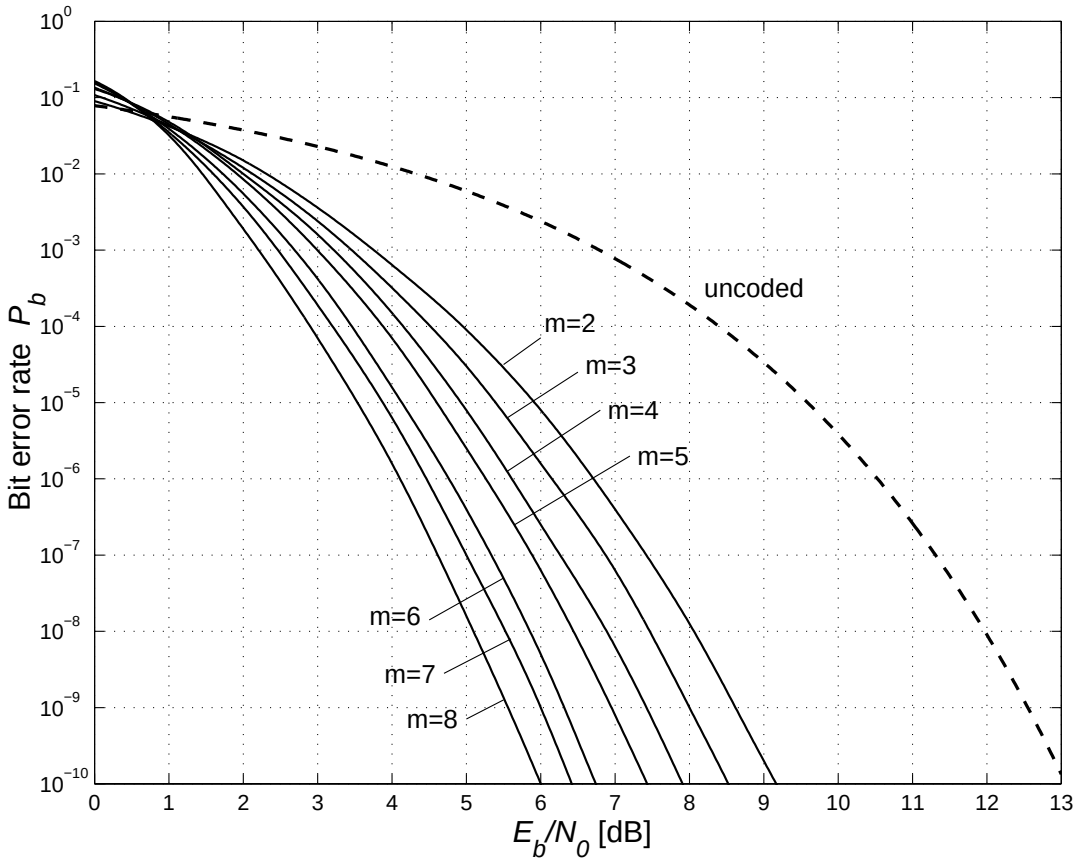

Figure 10.8. Bit-error probability of rate-1/2 codes with $m = 2, \ldots, 8$

gain von jeweils rund 0.4 dB. Bei $P_b = 10^{-5}$ beträgt der coding gain zwischen 3.7 dB für $m = 2$ und 6.1 dB für $m = 8$. Bei Hard-Decision verschieben sich die Kurven nach rechts, und zwar um ca. 2.2 dB bei moderaten Fehlerraten bzw. um 3 dB im asymptotischen Grenzfall. Bei oktaler Quantisierung mit 3 Bit verschieben sich die Kurven nur um ca. 0.2 dB nach rechts – aber die oktale Quantisierung setzt empfangsseitig eine einigermaßen genaue Pegelregelung voraus, so daß die Quantisierungsintervalle ähnlich wie in Bild 1.4 liegen.

In Figure 10.?ende erfolgt auch ein Vergleich zwischen soft- und hard-decision decoding für das Beispiel $R = 1/2$ mit $m = 6$.

In Bild 9.8 werden convolutional codes verschiedener Coderaten bei der Gedächtnislänge $m = 6$ miteinander verglichen, wobei die Coderaten $R = 2/3$ und $R = 3/4$ durch Punktierung aus $R = 1/2$ entstehen. Im Bereich größerer Fehlerraten ist eine kleine Coderate von Vorteil: So ist beispielsweise der $R = 1/3$-Code etwa 0.4 dB besser als der $R = 1/2$-Code, bei kleinen Fehlerraten reduziert sich allerdings dieser Vorsprung.

In Tabelle 9.2 sind die asymptotischen coding gains G_a einiger optimaler convolutional codes angegeben. Bei $m = 6$ ergibt sich beispielsweise ein Gewinn von rund 7 dB – unabhängig von der Coderate, d.h. die beiden Kurven für

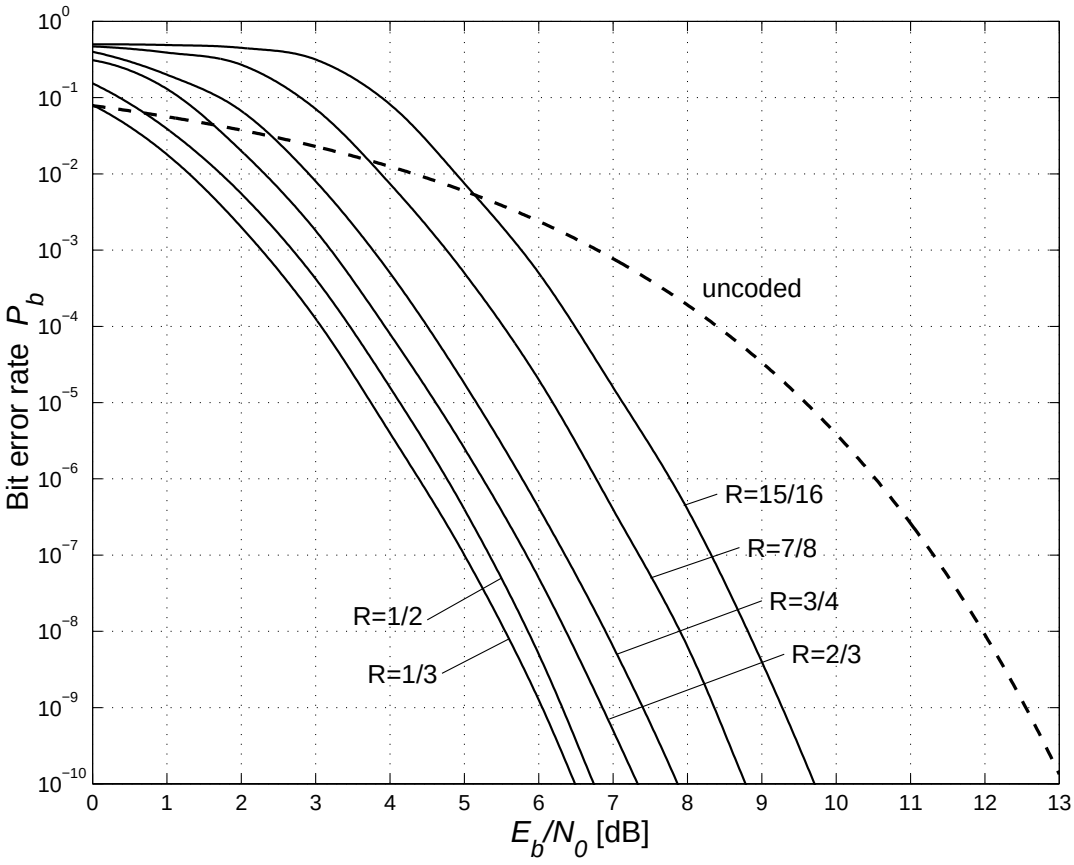

Figure 10.9. Bit-error probability of $m=6$ codes with $R = 1/3 \ldots 3/4$

Table 10.2. Asymptotic coding gains of optimum convolutional codes

m	$R = 1/2$			$R = 1/3$		
	d_f	$G_\mathrm{a,hard}$ [dB]	$G_\mathrm{a,soft}$ [dB]	d_f	$G_\mathrm{a,hard}$ [dB]	$G_\mathrm{a,soft}$ [dB]
2	5	0.97	3.98	8	1.25	4.26
3	6	1.76	4.77	10	2.22	5.23
4	7	2.43	5.44	12	3.01	6.02
5	8	3.01	6.02	13	3.36	6.37
6	10	3.98	6.99	15	3.98	6.99
7	10	3.98	6.99	16	4.26	7.27
8	12	4.77	7.78	18	4.77	7.78

$R = 1/2$ und $R = 1/3$ laufen in Bild 9.8 für sehr kleines P_b bzw. sehr großes E_b/N_0 zusammen.

In Tabelle 9.3 sind die coding gains bei moderaten Fehlerraten für die beiden Codes mit $R = 1/2$ und $R = 1/3$ bei $m = 6$ nochmals explizit angegeben. Die entsprechenden Werte sind näherungsweise auch aus Bild 9.8 ablesbar. Die kleinere Coderate hat also einen marginalen Vorteil ohne großen Mehraufwand bei der Decodierung, aber natürlich ist eine größere Bandbreite des AWGN

Table 10.3. Coding gains at moderate error rates (from [195])

P_b	Required E_b/N_0 [dB] uncoded	Coding gain [dB] with soft decisions, $m = 6$	
		$R = 1/2$	$R = 1/3$
10^{-3}	6.8	3.8	4.2
10^{-5}	9.6	5.1	5.7
10^{-7}	11.3	5.8	6.2
asymptotically		7.0	7.0

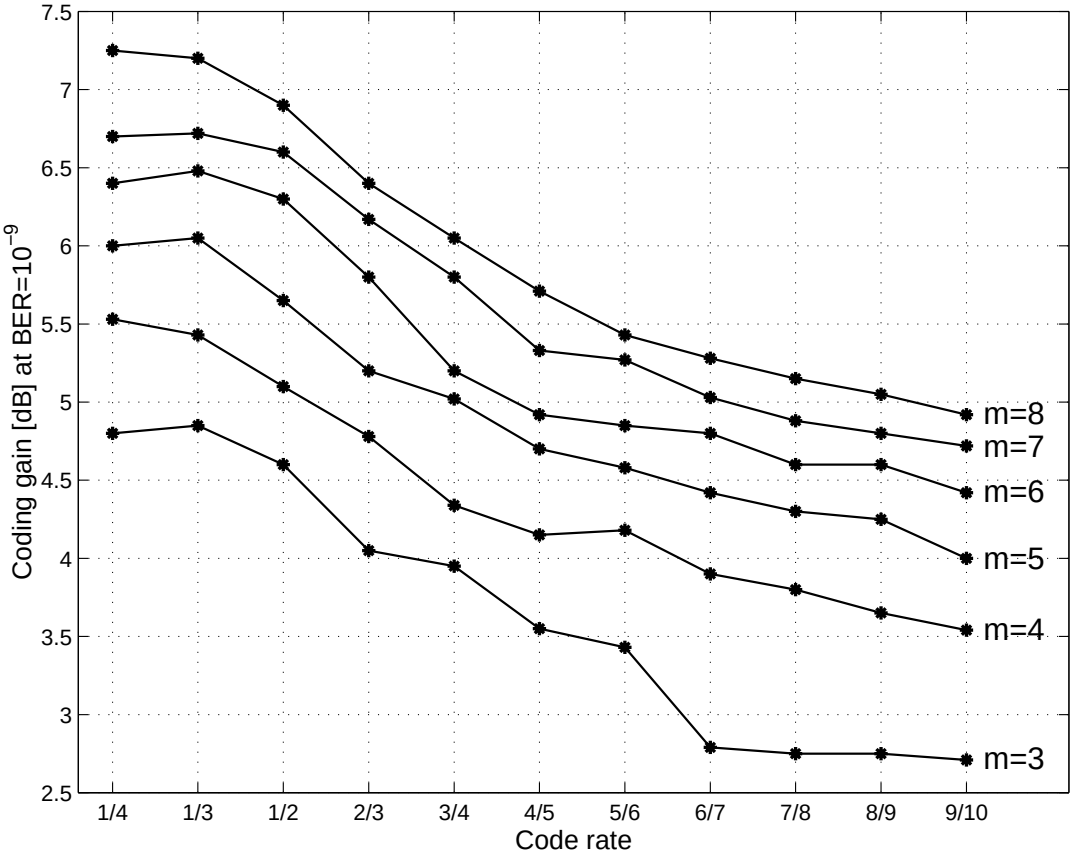

Figure 10.10. Coding gains for $R = 1/4 \ldots 9/10$ punctured convolutional codes with $m = 3 \ldots 8$ (from [199])

erforderlich.

In Bild 9.9 sind die coding gains G_{soft} bei idealer Soft-Decision für Codes mit den Gedächtnislängen $m = 3, \ldots, 8$ angegeben; im Gegensatz zu Tabelle 9.3 hier allerdings bei $P_b = 10^{-9}$. Coderaten größer als $1/2$ entstehen durch Punktierung aus $R = 1/2$. Offensichtlich ergeben sich durch Verkleinerung der Coderate unter $1/2$ keine wesentlichen Gewinne mehr, wie auch schon anhand von Bild 2.4 erklärt wurde. Dagegen sind auch bei hohen Coderaten noch erhebliche Gewinne zu erzielen, wobei erneut zu bemerken ist, daß die hohen Coderaten

fast ohne Mehraufwand in Encoder und Decoder realisierbar sind.

10.3.3 Viterbi Decoding for High-Level Modulation Schemes

Am Beispiel 64-QAM wird dargestellt, wie aus einem complex-valued received symbol six soft decisions für die six encoded bits eines symbols generiert werden. Zunächst entsprechen real and imaginary part of the received symbol jeweils 8-ASK modulation. Figure 10.10 zeigt wie aus einem real-valued 8-ASK symbol three soft decisions $s0, s1, s2$ generiert werden. Die gesendeten equispaced 8-ASK symbols werden mit $\mathcal{A}_{\text{mod,ASK}} = \{-7, -5, -3, -1, +1+3, +5, +7\}$ angenommen, und darüber ist das triplet $a1, a2, a3$ der encoded bits angegeben. Es gilt nun für y_I (und entsprechend y_Q)

$$\begin{aligned} s1 &= y_I \\ s2 &= 4 - |y_I| \\ s3 &= \left\{ \begin{array}{ll} |y_I| - 2 & \text{if } |y_I| < 4 \\ 6 - |y_I| & \text{if } |y_I| > 4 \end{array} \right\} \end{aligned} \tag{10.3.8}$$

Zur Vereinfachung der Implementierung kann man die Funktionen auch limitieren auf den range von -1 bis $+1$, ohne daß sich wesentliche Verschlechterungen ergeben. Wesentlich sind die Nulldurchgänge und das Vorzeichen der Steigungen (slopes).

Die

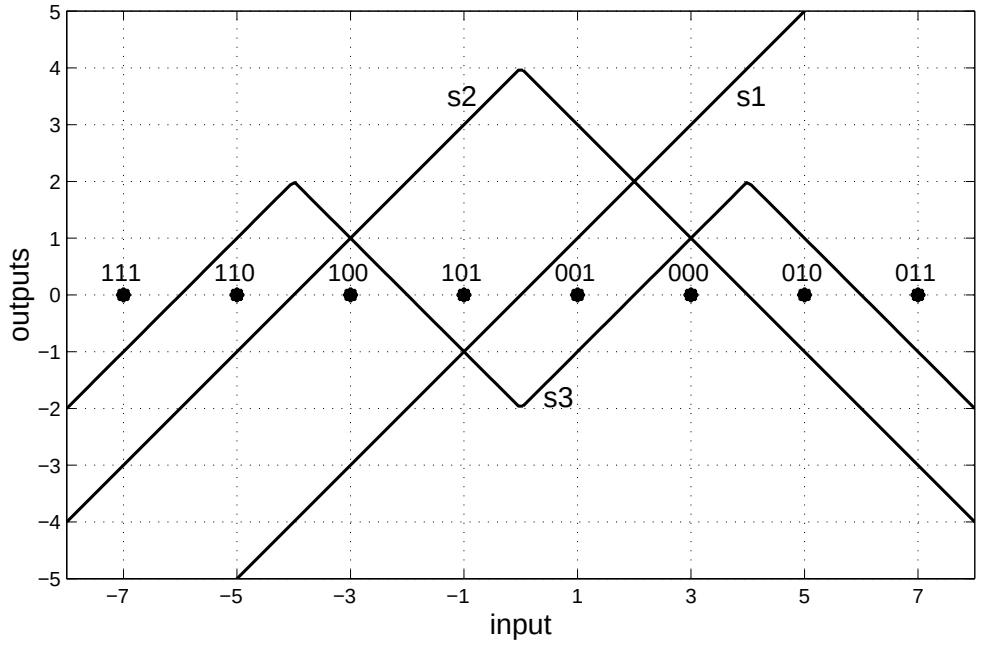

Figure 10.11. Generation of bit-wise soft decisions for Gray coded ASK

10.3.4 The Error Structure of Decoder Outputs

Wenn nach der ML-Decodierung von convolutional codes noch Fehler verbleiben, so treten diese Fehler gebündelt als burst errors auf. Durch das convolutional coding wird ein *super channel* or *outer channel* mit burst-error structure erzeugt, der aus dem convolutional encoder, dem mapping, dem inner waveform channel, dem demapping und dem convolutional decoder (siehe dazu auch Figure 9.11 and 12.?). Die Eigenschaften dieser burst error prägen den Entwurf von error-control schemes with concatenated codes ganz wesentlich und sind andererseits sehr bedeutsam für die data source bzw. den source decoder.

Die Figures 10.11 bis 10.14 zeigen die burst-error structure am Ausgang des convolutional decoders. Zum optischen Vergleich ist in Figure 10.11 zunächst ein pattern of the random single-errors of a BSC angegeben. Alle four Figures enthalten exakt 50×100 decoded bits, wobei ein cross einen error repräsentiert. Die mean error rate in all figures is approximately 0.04, thus every figure contains approximately 200 errors. Eine so hohe Fehlerrate haben wir gewählt um einen optisch eindrucksvollen Vergleich zu erhalten, obwohl das überhaupt nicht dem üblichen signal-to-noise ratio entspricht, bei dem convolutional codes üblicherweise betrieben werden.

Die von einem BSC erzeugten errors erscheinen in Figure 10.11 completely random und sind es auch tatsächlich auch. Dagegen ist die burst-error structure in den drei weiteren Figures sofort augenfällig, beim rate-1/2 code with $m = 2$ in Figure 10.12 ist die Länge eines bursts typischerweise auf 10 bis 15 information bits begrenzt, während es beim rate-1/2 code with $m = 6$ schon dutzende von bits sein können. Beim punctured rate-7/8-code with a puncturing period of $P = 7$ with a memory length of $m = 6$ treten kürzere burst errors als 50 bit scheinbar gar nicht mehr auf, dafür beeindruckt uns aber ein burst error mit fast 150 bit Länge.

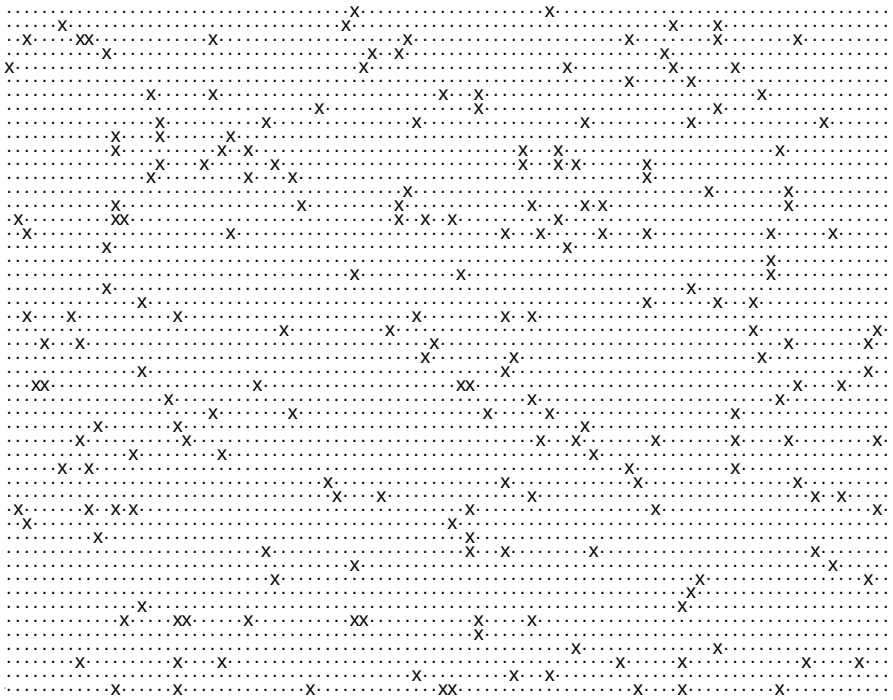

Figure 10.12. Random single errors of a BSC

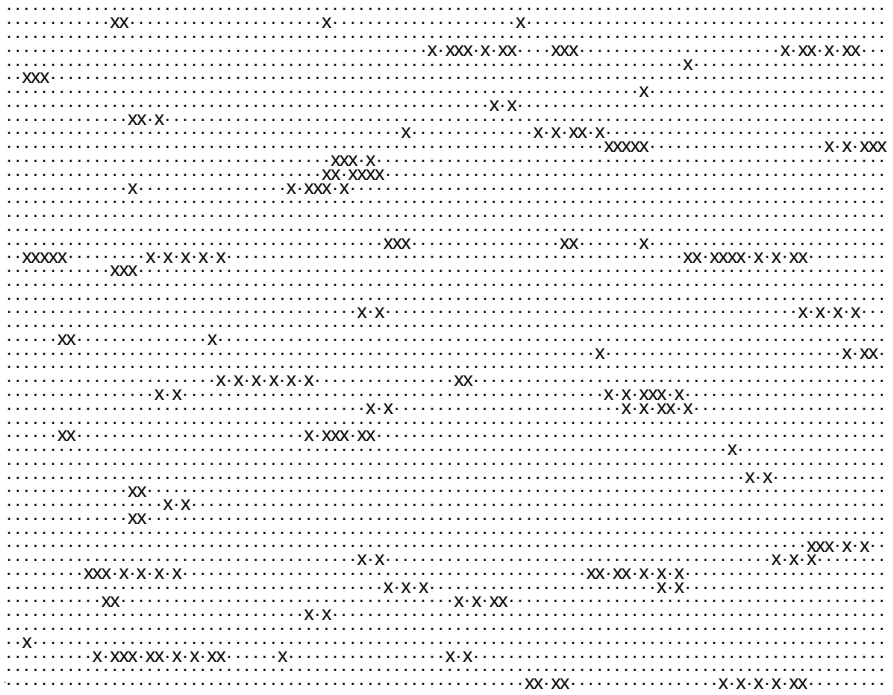

Figure 10.13. Burst errors at the Viterbi decoder output of a rate-1/2 code with $m = 2$ (at $E_b/N_0 = 1.1$ dB)

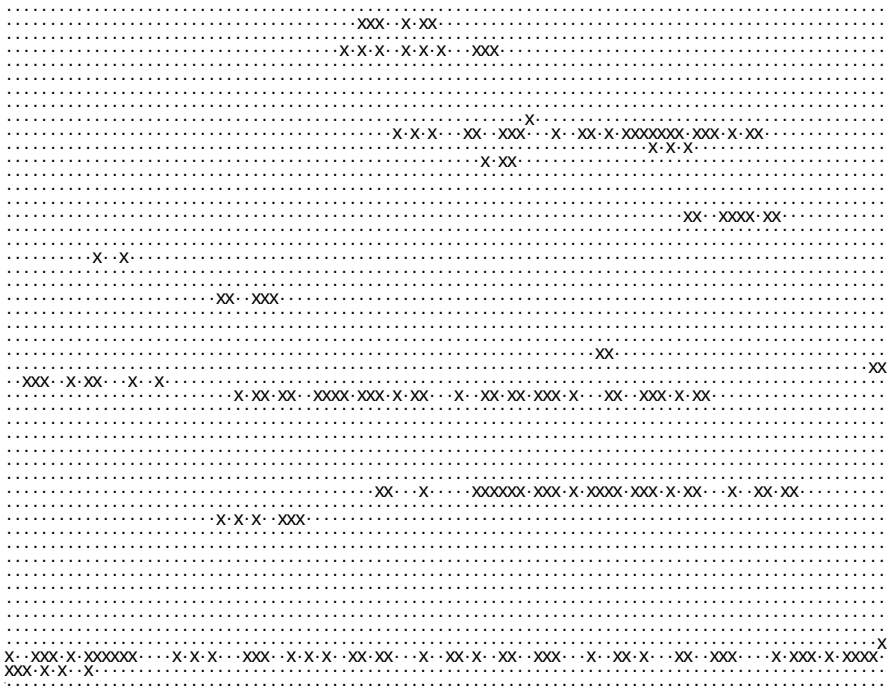

Figure 10.14. Burst errors at the Viterbi decoder output of a rate-1/2 code with $m = 6$ (at $E_b/N_0 = 1.1$ dB)

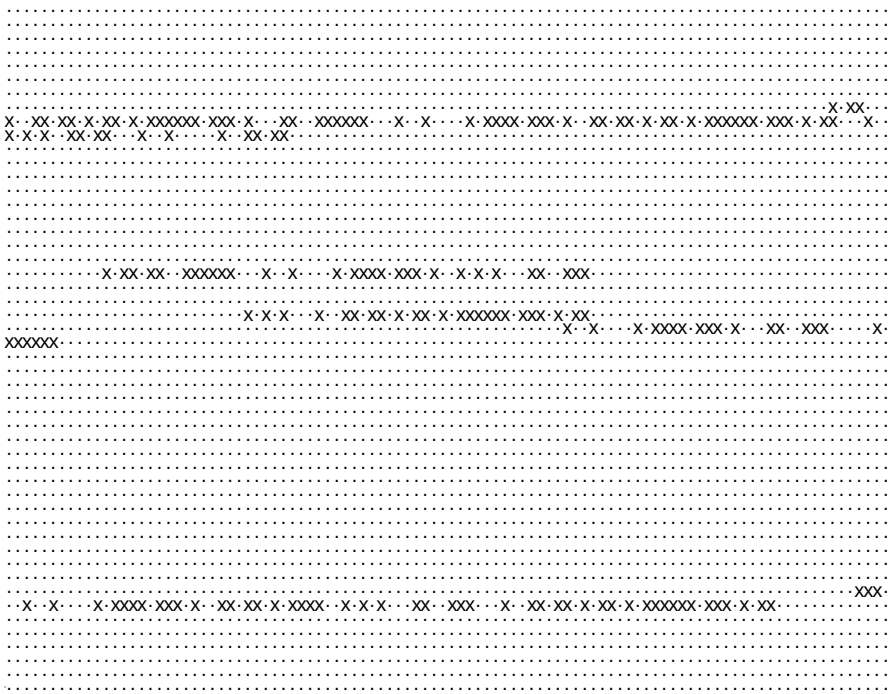

Figure 10.15. Burst errors at the Viterbi decoder output of a punctured rate-7/8 code with $m = 6$ (at $E_b/N_0 = 3.5$ dB)

Die Ursache für die burst-error structure kann leicht erklärt werden, denn ein nach der Decodierung noch verbleibendes error pattern besteht nach Section 10.? aus einer Sequenz von error events. Jedes error event ist mit mindestens d_f falschen encoded bits und mindestens einem falschen information bit verbunden. Typischerweise sind die error events aber länger mit mehreren information bit errors.

Offensichtlich nimmt die mittlere Länge von burst errors mit wachsender memory length zu. Eine weitere drastische Erhöhung ist bei punctured convolutional codes zu beobachten. Bei non-truncated punctured convolutional codes ist dann auch leicht einsehbar, daß das traceback memory möglich sehr groß sein sollte, für das rate-7/8 Beispiel sind 200 segments ausreichend, was weit oberhalb der in (10.?:?) formulierten rule of thumb von $v = 5m = 30$ liegt.

Für eine quantitative Beschreibung ist zunächst der Begriff eines burst errors präzise zu definieren. Ein burst error startet trivialerweise mit einem error und kann als beendet erklärt werden, wenn eine gewisse Anzahl Δ von folgenden information bits fehlerfrei ist. Die theoretische Analyse gestaltet sich am einfachsten, wenn ein Bündel durch $\Delta = m$ fehlerfreie information bits als beendet definiert wird. Dann kann eine Verteilung der burst-error lengths berechnet werden. Dabei zeigt sich das überraschende Ergebnis, daß die Häufigkeit eines burst errors nicht monoton mit der Länge des burst errors abnimmt. Investigations on the burst error statistics of Viterbi decoders can be found for example in [160, 167, 176, 208, 209].

Eine Anmerkung zur Simulation: Um eine bestimmte bit-error rate P_b nach dem decoder zu simulieren, sollten mindestens etwa 100 error simuliert werden (bei höheren Ansprüchen an die Genauigkeit wären auch 1000 errors oder noch mehr sinnvoll). Für $P_b = 10^{-5}$ wären beim BSC also etwa 10^7 information bits zu simulieren. Für den punctured convolutional code wären etwa 100 burst errors erforderlich, wenn jeder burst error im Mittel 50 error enthält, so sind also etwa 5000 errors und damit $5 \cdot 10^8$ information bit erforderlich. Punctured codes sind also wesentlich aufwendiger zu simulieren als non-punctured codes with short memory oder gar uncoded modulation systems.

10.4 Concatenated Codes and Requirements on Soft-Decision Output

Ein Codierungssystem mit *verketteten Codes* (concatenated coding) zeigt Bild 9.11. Dabei werden zwei Codes in Reihe geschaltet, d.h. die von einem ersten (äußeren) Encoder erzeugte Symbolfolge wird in einem zweiten (inneren) Encoder mit zusätzlicher Redundanz versehen.

Typische Anwendungen mit verketteten Codes nennt Tabelle 9.4, die in den nachfolgenden Kapiteln teilweise noch ausführlich behandelt werden. Die Codeverkettung ist übrigens nicht unbedingt auf zwei Stufen beschränkt. Das Prinzip

der verketteten Codierung wird an dieser Stelle nur deshalb schon eingeführt, damit eine gewisse Anforderung an den inneren Decoder hergeleitet werden kann.

Table 10.4. Typical applications of concatenated coding

	Innerer Code	Äußerer Code	siehe Abschnitt
1	block code	block code	11.9, 12.6
2	convolutional code	RS-Code	12.1
3	MLSE-Entzerrer	convolutional code, TCM	11.5, 12.3, 12.4
4	convolutional code	Quellencode	12.3, 12.4
5	convolutional code	convolutional code	(12.1)

Durch das Interleaving-Verfahren in Bild 9.11 können die Bündelfehler am Ausgang des inneren Decoders in Pseudo-Einzelfehler überführt werden. Dazu wird im Interleaver die Symbolfolge bzw. Bitfolge umsortiert. Im Deinterleaver wird diese Umsortierung wieder rückgängig gemacht, so daß das gesamte System Interleaver-Deinterleaver lediglich eine Verzögerung bewirkt und ansonsten transparent ist. Geeignete Methoden für Interleaving werden in Abschnitt 11.1 behandelt.

Durch das Interleaving kann also ein quasi gedächtnisloser Super-Kanal erzeugt werden, der natürlich aufgrund des inneren Decoders ein Hard-Decision Kanal ist. Falls jedoch für den äußeren Decoder Soft-Decision verfügbar wäre, könnten damit erhebliche Gewinne erzielt werden. Bei einem AWGN-Kanal können diese Gewinne sogar indirekt in E_b/N_0 umgerechnet werden, d.h. die Kanalcodierung verbessert nicht nicht nur die Fehlerrate, sondern direkt den Signal/Rausch-Abstand des Kanals [185, 187]. In jedem Fall wird aber die Fehlerrate nach dem äußeren Decoder bei einem Super-Kanal mit Soft-Decision

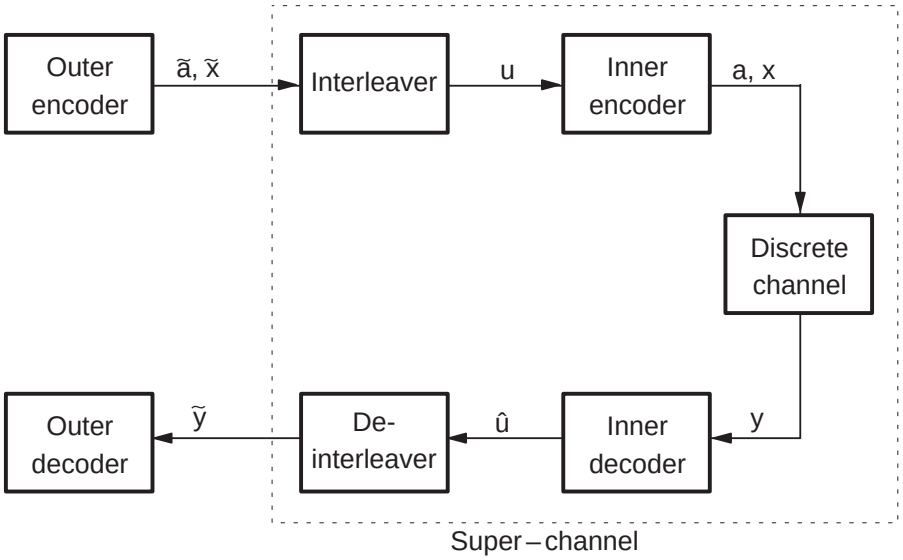

Figure 10.16. Concatenated coding

erheblich niedriger ausfallen.

Ziel ist es also, den Soft-Decision-Output des inneren Decoders als Soft-Decision-Input für den äußeren Decoder zu verwenden. Denn es ist anschaulich sofort klar, daß Verbesserungen zu erwarten sind, wenn der äußere Decoder nicht nur die harten Entscheidungen des inneren Decoders erhält, sondern zusätzlich Zuverlässigkeitsinformationen, d.h. ein Maß dafür, wie sicher die einzelnen Entscheidungen des inneren Decoders sind. Wenn der innere Decoder seine eigenen einzelnen Entscheidungen beispielsweise als sehr sicher bzw. als sehr unsicher einstuft, so sollten diese Entscheidungen im äußeren Decoder mit großem Gewicht eingehen bzw. als Ausfälle ignoriert werden.

Im Interleaver werden also Symbole oder Bits verarbeitet, im Deinterleaver dagegen weiche Entscheidungen bzw. reelle Zahlen, die für die numerische Verarbeitung natürlich einer gewissen Quantisierung unterliegen.

In diesem Abschnitt werden die Anforderungen an einen vernünftigen Soft-Decision-Output formuliert und im nächsten Abschnitt wird dann die praktische Berechnung dieses Outputs in einem erweiterten Viterbi-Algorithmus vorgestellt. Die grundlegende Idee ist, daß die Zuverlässigkeit der Entscheidungen des inneren Decoders auch bei einem zeitinvarianten diskreten Kanal als von Bit zu Bit schwankend angesehen wird – und zwar in Abhängigkeit von den vorangehenden und nachfolgenden Empfangswerten y. Bei einem zeitvarianten diskreten Kanal wird damit auch die schwankende Güte der Übertragung automatisch in richtiger Weise berücksichtigt.

Der Super-Kanal wird als zeitvariant, gedächtnislos, symmetrisch und binär mit $\mathcal{A}_{\text{in}} = \mathcal{A}_{\text{out}} = \{+1, -1\}$ angenommen. Input und Output werden wie in Bild 9.11 mit $\tilde{x}$ bzw. $\tilde{y}$ bezeichnet. Für die Übergangswahrscheinlichkeit gilt:

$$P(\hat{u}_r|u_r) = P(\tilde{y}_r|\tilde{x}_r) = \left\{ \begin{array}{ll} 1-p_r & \tilde{x}_r = \tilde{y}_r \\ p_r & \tilde{x}_r \neq \tilde{y}_r \end{array} \right\}. \tag{10.4.1}$$

Also ist $p_r = P(\hat{u}_r \neq u_r) = P(\tilde{y}_r \neq \tilde{x}_r)$ die zeitvariante Bit-Fehlerwahrscheinlichkeit. Ähnlich wie bei Example 9.1 gilt für die Viterbi-Metrik des Super-Kanals:

$$\mu(\tilde{y}_r|\tilde{x}_r) = \alpha \ln P(\tilde{y}_r|\tilde{x}_r) + \beta_r = \alpha \left\{ \begin{array}{ll} \ln(1-p_r) & \tilde{y}_r = \tilde{x}_r \\ \ln p_r & \tilde{y}_r \neq \tilde{x}_r \end{array} \right\} + \beta_r.$$

Mit $\alpha = 2$ und $\beta_r = -2 \ln p_r - \ln((1-p_r)/p_r)$ folgt:

$$\begin{aligned} \mu(\tilde{y}_r|\tilde{x}_r) &= \left\{ \begin{array}{ll} 2 \ln \dfrac{1-p_r}{p_r} & \tilde{y}_r = \tilde{x}_r \\ 0 & \tilde{y}_r \neq \tilde{x}_r \end{array} \right\} - \ln \frac{1-p_r}{p_r} \\ &= \left\{ \begin{array}{ll} +\ln \dfrac{1-p_r}{p_r} & \tilde{y}_r = \tilde{x}_r \\ -\ln \dfrac{1-p_r}{p_r} & \tilde{y}_r \neq \tilde{x}_r \end{array} \right\} \\ &= \tilde{x}_r \tilde{y}_r \ln \frac{1-p_r}{p_r}. \end{aligned}$$

Für den inneren Kanal (von u nach $\hat{u}$) gilt entsprechend

$$\mu(\hat{u}_r|u_r) \;=\; u_r \cdot \underbrace{\hat{u}_r \ln \frac{1-p_r}{p_r}}_{=\,w_r}. \tag{10.4.2}$$

Zur Erinnerung: Sowohl für den zeitinvarianten AWGN wie für den zeitinvarianten BSC lautet die Viterbi-Metrik $\mu(y_i|x_i) = x_i y_i$ für den inneren Decoder, was sich auch als Spezialfall von (9.7.2) bei konstantem p_r ergibt. Wenn statt $\hat{u}_r$ nun w_r als Output des Super-Kanals angesehen wird, dann hat die Viterbi-Metrik $\mu(\hat{u}_r|u_r) = \mu(w_r|u_r) = u_r w_r$ für den äußeren Decoder die gleiche Produktform wie für den inneren Decoder. Folglich verarbeiten innerer und äußerer Decoder Soft-Decision-Input identisch. Der innere Decoder sollte zusätzlich den Soft-Decision-Output

$$w_r \;=\; \hat{u}_r \cdot \ln \frac{1-p_r}{p_r} \;=\; \hat{u}_r \cdot \ln \frac{P(\hat{u}_r = u_r)}{P(\hat{u}_r \neq u_r)} \tag{10.4.3}$$

berechnen und an den äußeren Decoder abgeben. Dieses Resultat sieht etwas seltsam aus, aber die beiden Feststellungen

$$\begin{aligned} p_r < 0.5 \;&\Longrightarrow\; \mathrm{sign}(w_r) = \hat{u}_r \\ p_r \text{ klein} \;&\Longrightarrow\; |w_r| \text{ groß} \end{aligned}$$

erscheinen sehr vernünftig. Punktierte Stellen bei convolutional codes können entweder über $\hat{u}_r = 0$ bzw. $\tilde{y}_r = 0$ oder über $p_r = 0.5$ eingerechnet werden. Die Aufgabe für den inneren ML-Decoder kann nun wie folgt formuliert werden: Neben den harten Entscheidungen $\hat{u}_r$ sollte auch das Zuverlässigkeitsmaß

$$L_r \;=\; \ln \frac{1-p_r}{p_r} \;=\; \ln \frac{P(\hat{u}_r = u_r)}{P(\hat{u}_r \neq u_r)} \tag{10.4.4}$$

berechnet werden, und anstelle dieser beiden einzelnen Größen benötigt der äußere Decoder nur das Produkt $w_r = \hat{u}_r L_r$.

10.5 The Soft-Output Viterbi Algorithm (SOVA)

Der Soft-Decision-Output $L_r = \ln((1-p_r)/p_r)$ kann mit einer Erweiterung des Viterbi-Algorithmus näherungsweise relativ einfach berechnet werden. Ein entsprechendes Verfahren wurde mit der Abkürzung SOVA (Soft-Output Viterbi-Algorithmus) von J.Hagenauer [185] eingeführt. Es gibt allerdings noch eine ganze Reihe anderer Verfahren, die nicht nur auf rein anschaulichen Begründungen fußen, sondern auch das theoretisch exakte Ergebnis liefern, aber

komplizierter in der Realisierung sind und nicht als Erweiterung des VA interpretiert werden können. Ein solches Verfahren wird inzwischen als SDMAPA (Soft-Deciding Symbol-by-Symbol Maximum Aposteriori Algorithm) bezeichnet [185, 212].

Zunächst wird jeder der 2^m Zustände z_{r+1} betrachtet, bei dem jeweils zwei Wege anliegen. Es sei W_1 der Survivor und W_2 der nicht überlebende Weg, d.h.

$$P(\boldsymbol{y}|W_1) \geq P(\boldsymbol{y}|W_2). \tag{10.5.1}$$

Die Wahrscheinlichkeit dafür, daß W_1 nicht der Survivor ist, berechnet sich zu

$$q(z_{r+1}) = P(W_2|\boldsymbol{y}) = \frac{P(\boldsymbol{y}|W_2)P(W_2)}{P(\boldsymbol{y}|W_1)P(W_1) + P(\boldsymbol{y}|W_2)P(W_2)}.$$

Bei gleichen Apriori-Wahrscheinlichkeiten sind alle Wege gleichwahrscheinlich:

$$q(z_{r+1}) \;=\; \frac{P(\boldsymbol{y}|W_2)}{P(\boldsymbol{y}|W_1) + P(\boldsymbol{y}|W_2)} \;=\; \frac{1}{1 + \dfrac{P(\boldsymbol{y}|W_1)}{P(\boldsymbol{y}|W_2)}}. \tag{10.5.2}$$

Bei $P(\boldsymbol{y}|W_1) \gg P(\boldsymbol{y}|W_2)$ ist die Entscheidung auf W_1 sehr sicher und folglich gilt $q(z_{r+1}) \approx 0$. Bei $P(\boldsymbol{y}|W_1) \approx P(\boldsymbol{y}|W_2)$ ist die Entscheidung auf W_1 ziemlich unsicher und folglich gilt $q(z_{r+1}) \approx 0.5$. Bei einer Normierung auf $\mathcal{A}_{\text{in}} = \{-1, +1\}$ gilt $E(y^2) = 1 + N_0/(2E_c)$ und die Viterbi-Metrik zum Weg W bzw. zur Sendefolge $\boldsymbol{a}$ hat die Form

$$\begin{aligned} \mu(\boldsymbol{y}|W) \;&=\; \ln P(\boldsymbol{y}|W) + \beta \;=\; -\frac{E_c}{N_0}\|\boldsymbol{y} - \boldsymbol{a}\|^2 + \beta \\ &=\; 2\frac{E_c}{N_0} \sum_{i \leq r+1} \sum_{\nu=1}^{n} y_{i,\nu} a_{i,\nu}. \end{aligned} \tag{10.5.3}$$

Die Metrik kann im Gegensatz zum Standard-VA nicht skaliert werden, d.h. E_c/N_0 muß zumindest näherungsweise bekannt sein, da gemäß (9.8.2) folgende Darstellung angestrebt wird:

$$q(z_{r+1}) = \frac{1}{1 + e^{\Delta}} \quad \text{mit} \quad \Delta = \mu(\boldsymbol{y}|W_1) - \mu(\boldsymbol{y}|W_2) \geq 0. \tag{10.5.4}$$

Mit $u_{0,W}, \ldots, u_{r,W}$ werden die information bits zum Weg W nach z_{r+1} bezeichnet. Die Fehlerwahrscheinlichkeit für $u_{r,W}$ wird als

$$p_r = P(u_{r,W} \text{ ist falsch}) \tag{10.5.5}$$

geschrieben. Gesucht ist $L_r = \ln((1 - p_r)/p_r)$ zum endgültigen Survivor. Für die beiden Wege W_1, W_2 nach $z_{r+1} = (u_r, \ldots, u_{r-m+1})$ gilt $u_{r,W_1} = u_{r,W_2}$ und somit kann hier kein Fehler passieren, d.h. $p_r = 0$.

Bei $i < r$ und $u_{i,W_1} = u_{i,W_2}$ bleibt p_i ungeändert, auch wenn W_1 nicht der richtige Weg ist.

Bei $i < r$ und $u_{i,W_1} \neq u_{i,W_2}$ ändert sich p_i durch die Verlängerung der Wege und wird rekursiv aktualisiert: Anschaulich begründbar sind zwei Effekte: Mit der Wahrscheinlichkeit $1 - q(z_{r+1})$ ist W_1 der richtige Weg und somit reduziert sich p_i um diesen Faktor. Mit der Wahrscheinlichkeit $q(z_{r+1})$ ist W_1 der falsche Weg und dann ist u_{i,W_1}=falsch gleichbedeutend mit u_{i,W_2}=richtig. Insgesamt folgt:

$$p_i^{\text{neu}} = \left\{ \begin{array}{ll} 0 & i = r \\ p_i & i < r,\ u_{i,W_1} = u_{i,W_2} \\ p_i(1 - q(z_{r+1})) + (1 - p_i)q(z_{r+1}) & i < r,\ u_{i,W_1} \neq u_{i,W_2} \end{array} \right\}. \quad (10.5.6)$$

Im dritten Fall gilt $p_i^{\text{neu}} = p_i(1-2q(z_{r+1}))+q(z_{r+1})$ und damit folgt aus $0 \leq p_i \leq 1/2$ sofort $0 \leq p_i^{\text{neu}} \leq 1/2$. Für $L_i = \ln((1-p_i)/p_i)$ ergibt sich unter Beachtung von

$$e^{L_i} = \frac{1 - p_i}{p_i} \quad , \quad e^{\Delta} = \frac{1 - q(z_{r+1})}{q(z_{r+1})}$$

für den dritten Fall aus (9.8.6) mit einfacher Rechnung:

$$\begin{aligned} L_i^{\text{neu}} &= \ln \frac{1 - p_i^{\text{neu}}}{p_i^{\text{neu}}} = \ln \frac{1 - p_i(1 - q(z_{r+1})) - (1 - p_i)q(z_{r+1})}{p_i(1 - q(z_{r+1})) + (1 - p_i)q(z_{r+1})} \\ &= \ln \frac{p_i q(z_{r+1}) \left(1 + \dfrac{1 - p_i}{p_i} \cdot \dfrac{1 - q(z_{r+1})}{q(z_{r+1})}\right)}{p_i q(z_{r+1}) \left(\dfrac{1 - p_i}{p_i} + \dfrac{1 - q(z_{r+1})}{q(z_{r+1})}\right)} = \ln \frac{1 + e^{L_i + \Delta}}{e^{L_i} + e^{\Delta}}. \quad (10.5.7) \end{aligned}$$

Für die Funktion aus (9.8.7) sind folgende Darstellungsformen bekannt [189] (Erinnerung: $\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$, $\operatorname{artanh}(x) = \ln \dfrac{1 + x}{1 - x}$):

$$\begin{aligned} \ln \frac{1 + e^{u+v}}{e^u + e^v} &= \ln \frac{1 + \tanh(u/2)\tanh(v/2)}{1 - \tanh(u/2)\tanh(v/2)} & (10.5.8) \\ &= \operatorname{artanh}(\tanh(u/2)\tanh(v/2)). & (10.5.9) \end{aligned}$$

Neben einer Berechnung oder Tabellierung dieser Funktion kann auch die Approximation

$$\ln \frac{1 + e^{u+v}}{e^u + e^v} \approx \operatorname{sign}(u)\operatorname{sign}(v) \cdot \min\{|u|, |v|\} \quad (10.5.10)$$

verwendet werden, mit der sich aus (9.8.6) und (9.8.7) das gesamte Verfahren wie folgt ergibt:

$$L_i^{\text{neu}} = \left\{ \begin{array}{ll} \infty & i = r \\ L_i & i < r,\ u_{i,W_1} = u_{i,W_2} \\ \min\{L_i, \Delta\} & i < r,\ u_{i,W_1} \neq u_{i,W_2} \end{array} \right\}. \quad (10.5.11)$$

Das SOVA-Verfahren ist eine Erweiterung des normalen Viterbi-Algorithmus ohne Änderungen, d.h. zwischen Normal- und SOVA-Betrieb kann simpel

umgeschaltet werden. Bei jedem der 2^m Zustände wird der Survivor W_1 und die Metrikdifferenz Δ gemäß (9.8.4) bestimmt. Der Survivor wird zurückverfolgt und die L_i werden entsprechend (9.8.11) aktualisiert. Neben den Survivor-Wegen (beschrieben durch die $u_{i,W}$) müssen auch die L_i abgespeichert werden. Die beim endgültigen Survivor verbleibenden L_i bilden dann den Soft-Decision-Output. Gegenüber dem standardmäßigen Viterbi-Algorithmus erfordert der SOVA einen Mehraufwand von etwa 60 bis 80% [187, 188, 189].

Wenn der diskrete Kanal ein AWGN-Kanal ist, kann der Super-Kanal aus Bild 9.11 näherungsweise ebenfalls als AWGN-Kanal aufgefaßt werden, indem der SOVA als Filter interpretiert wird. In [185, 187] wird der Zusammenhang zwischen den Signal/Rausch-Abständen des inneren diskreten AWGN-Kanals und des äußeren Super-Kanals untersucht.

10.6 Comparison of Block and Convolutional Codes

In Tabelle 9.6 erfolgt eine ausführliche Gegenüberstellung von block codes und convolutional codes, die sich auf die "Normalfälle" bezieht. Für spezielle Anwendungen sind inzwischen Hunderte oder Tausende von spezifischen Codierungssystemen entwickelt worden. Dabei trifft der eine oder andere Punkt nicht immer in voller Allgemeinheit zu, aber primär soll die Tabelle nur eine grobe Orientierung vermitteln. Einige Aussagen werden erst anhand der folgenden Kapitel verständlich, das gilt insbesondere für die trelliscodierte Modulation (TCM) aus Kapitel 10 sowie für verschiedene Anwendungen und Ergänzungen aus den Kapiteln 11 und 12.

Um die Angaben der Tabelle 9.6 für den AWGN-Kanal mit und ohne harter Quantisierung numerisch zu illustrieren, erfolgt ein quantitativer Vergleich zwischen einem BCH-Code und einem convolutional code bei fast gleicher Coderate. Aus den Bildern 7.1 und 9.7 ergeben sich die in Tabelle 9.5 aufgeführten Werte:

Table 10.5. Bit-error probability of BCH and convolutional codes

P_b	Notwendiges E_b/N_0 [dB] für P_b		
	$(255, 131)$-BCH-Code $d_{\min} = 37$, BM-Decoder	$R = 1/2$-convolutional code mit $m = 6$ $d_{\mathrm{f}} = 10$, ML-Decoder	
	Hard-Dec.	Hard-Dec.	Soft-Dec.
10^{-3}	4.9	4.7	2.6
10^{-6}	6.0	7.0	4.8
10^{-10}	7.0	8.9	6.7
G_{a}	9.9	4.0	7.0

Bei Hard-Decision Decodierung ist der convolutional code lediglich bei einem schlechten Kanal etwas besser als der block code, bei einem guten Kanal ist

dagegen der block code aufgrund der höheren Minimaldistanz besser als der convolutional code, nämlich um 1.9 dB bei $P_b = 10^{-10}$ und asymptotisch sogar um 5.9 dB.

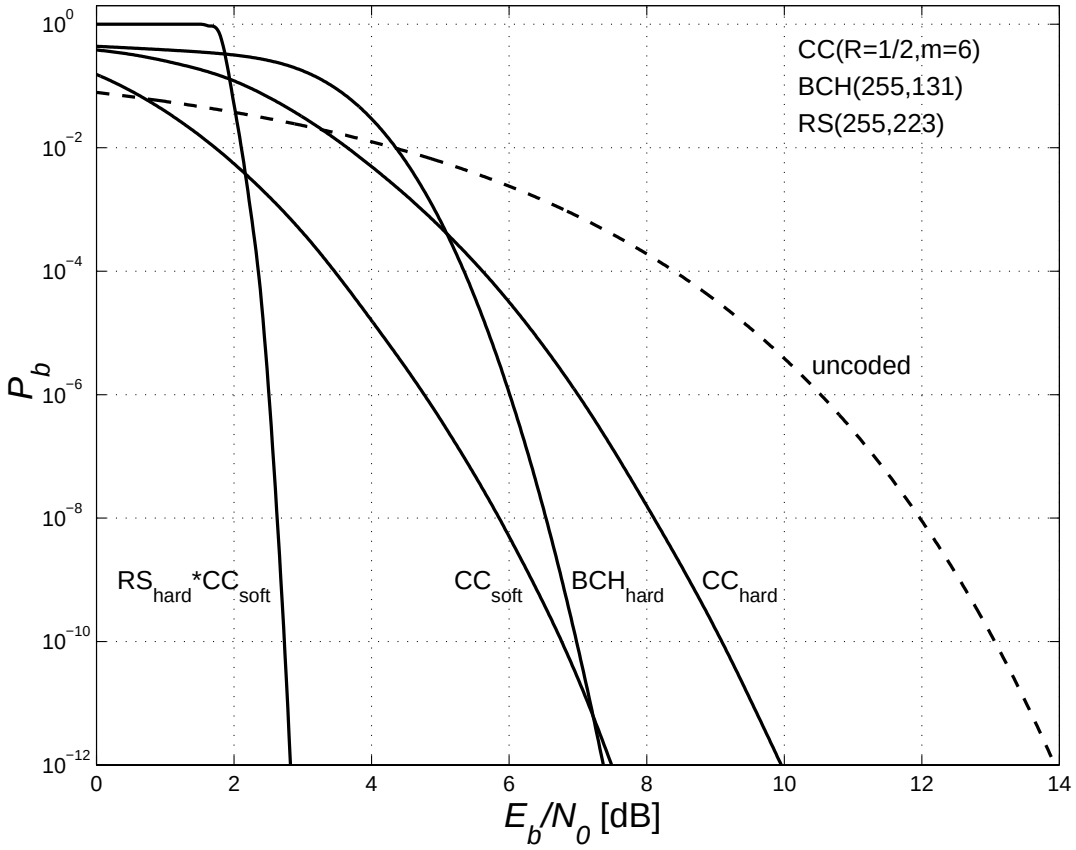

Figure 10.17. Performance comparison of block and convolutional codes

Bei Soft-Decision Decodierung für den convolutional code muss für den BCH-Code realistischerweise weiterhin eine Hard-Decision Decodierung nach dem BMD-Prinzip angenommen werden. Dann ist der convolutional code über den ganzen Bereich besser, und zwar um 2.3 dB bei $P_b = 10^{-3}$ und um 0.3 dB bei $P_b = 10^{-10}$. Lediglich asymptotisch ist der BCH-Code um 2.9 dB besser.

In der Decodierverzögerung ist der convolutional code mit ca. 32 information bits besser als der BCH-Code mit 131 information bits.

Wenn die Korrekturfähigkeit der Codes durch zu viele Fehler überschritten wird, offenbart sich ein wesentlicher Unterschied zwischen convolutional - und block codes: Der Viterbi-Decoder produziert burst errors, während die RS- und BCH-BMD-Decoder in den meisten Fällen ein unkorrigierbares error pattern signalisieren und nur selten eine falsche Korrektur ausführen.

Die convolutional codes sind den block codes vor allem dann überlegen, wenn ein Kanal mit schlechter bis mittelmäßiger Qualität um zwei oder drei Zehnerpotenzen in der Fehlerrate verbessert werden soll. Wenn dagegen extrem

kleine error rates angestrebt werden, sind BCH- und RS-Codes vorzuziehen. Beide Codeklassen werden aber übertroffen durch die concatenation of block and convolutional codes, die in Abschnitt 12.1 behandelt wird.

Viele Aspekte sind zu beachten: speech codecs are robust to moderate channel errors, but cannot tolerate long transmission delays. RS codes offer the feature of good error detection in addition to error correction, which could be quite useful for ARQ schemes.

Table 10.6. General comparison: block codes versus convolutional codes

Kriterium	block codes	convolutional codes
Historie	zu Beginn der Informationstheorie überwiegend verwendet, Beweis der coding theorems	werden inzwischen gleichoft verwendet wie block codes
Mathematik	weit entwickelte und geschlossene Theorie, teilweise sehr anspruchsvoll	Theorie mit der gleichen Aussagekraft wie bei block codes ist noch schwieriger, wird aber selten benötigt
$\mathbb{F}_{p^m}$	$p = 2$ (in Sonderfällen $p > 2$) $m \leq 10$ for RS codes $m \leq 10 \ldots 14$ for BCH codes	$p = 2$ $m = 1$
Grundstruktur	linear spaces (Matrizen über Körpern): Intensive Kenntnisse der Galois field-Arithmetik erforderlich bei Entwicklung und Implementierung hochentwickelter Verfahren	Module?? (Matrizen über polynomial rings): Intensive Kenntnisse of module?? theory erforderlich bei Umformung von (n, k)-Encodern, aber praktisch selten notwendig
Beziehung zueinander (formal)	block codes sind spezielle convolutional codes without memory	convolutional codes are generalized block codes und truncated convolutional codes sind spezielle block codes
Code rate	$R = k/n$ nahezu beliebig zwischen 0 und 1	$R = 1/n$, $R \to 1$ mit Punktierung
Gute Codes durch	große block length n	große memory length m
Leistungsfähige Codes	können als parametrische Klasse analytisch geschlossen angegeben werden (RS, BCH)	werden durch Rechnersuche gewonnen, sind nicht theoretisch ableitbar
Gütekriterium	minimum distance, weight distribution	freie Distanz, weight distribution
Polynomial description	1 generator polynomial vom Grad $n - k$ (bei zyklischen Codes)	$k \cdot n$ generator polynomial vom Grad $\leq m$ (typisch $k = 1$)
weitere Charakterisierungen	spektral (DFT)	trellis, state, tree diagram
Systematische Codes	sind immer möglich und werden immer verwendet	haben generelle Nachteile und werden nur in Sonderfällen verwendet (Ausnahme: rückgekoppelte Encoder bei TCM)

Fortsetzung von Tabelle 9.6

Kriterium	block codes	convolutional codes
Fehlerfort-pflanzung	entfällt	möglich, entschärft bei Terminierung
Decodierung	hochentwickelte algebraische Decodierverfahren (BMA,EA) für komplexe Codes als BMD	anschaulich begründete Verfahren (VA) als MLD, sowie sequentielle und algebraische Verfahren
Soft-Decision	nur bei einfachen Codes praktikabel, ansonsten nur Ausfalldecodierung	möglich mit geringem Mehr-aufwand, 2 bis 3 dB Gewinn, Soft-Output möglich
Beurteilung des Modulations-systems	mit der Fehlerwahrschein-keit (Hard-Decision)	mit dem R_0-Kriterium, "Fehlerkorektur" findet nicht statt (Soft-Decision)
Fehlererken-nung ohne Korrektur	gut möglich, zyklische Codes sind besonders geeignet zur Erkennung von Bündelfehlern, ARQ-Verfahren	meist unmöglich
Synchronisation	erforderlich, teilweise das größte Problem überhaupt	VA ist selbstsynchronisierend, schnelle Träger- und Phasen-synchronisation bei rotationsinvarianter TCM
Hauptanwen-dungen	reine Binärkanäle mit und ohne Bündelstörungen	für reine Binärkanäle weniger gut geeignet
	sehr gut anpaßbar an Bündelstörungen	Bündelstörungen erfordern Interleaving, mit Mehraufwand anpaßbar
	für AWGN-Kanäle mit Soft-Decision weniger gut geeignet	für AWGN-Kanäle mit Soft-Decision bestens geeignet, für Kanäle mit CSI
		in Kombination mit Quellencodierung (RCPC-Codes, quellen-gesteuerte Decodierung)
	wenn Daten bereits in Block-form vorliegen und nur die Blockfehlerrate wesentlich ist	
	blockcodierte Modulation (BCM)	trelliscodierte Modulation (TCM)
	Verkette Codes Fadingkanäle (Rayleigh, Rice) bandbegrenzte Kanäle Kanäle mit Verzerrungen UEP-Anwendungen	

Beim Vergleich der verfügbaren Hardware-Implementierungen zeigt sich, daß die erreichbaren Datenraten bei RS and BCH decoders with a block length of 255 etwa um das 2- bis 5-fache höher ausfallen als bei einem Viterbi decoder with 64 states.

Praktische Aspekte sehr steiler Kurven.....

10.7 Problems

10.1. Betrachte das Standardbeispiel $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$ mit Terminierung und der fehlerfreien Empfangsfolge (Hard-Decision)

$$\boldsymbol{y} = 11\ 01\ 01\ 00\ 10\ 11.$$

Der Anfangszustand soll als unbekannt angenommen werden und alle Survivor-Metriken sind auf den gleichen Wert vorbelegt: $\mu_0(i) = 0$. Führe die Viterbi-Decodierung durch.

Was passiert bei einem non-truncated Code, wenn bei z_6 mit der Best State Rule bzw. mit der Zero State Rule entschieden wird?

10.2. Betrachte das Standardbeispiel $\boldsymbol{G}(x) = (1+x+x^2, 1+x^2)$ mit truncation. Die Sendefolge

$$\boldsymbol{a} = 11\ 01\ 01\ 00\ 10\ 11$$

wird verfälscht zu den beiden fehlerhaften received sequences (Hard-Decision)

$$\boldsymbol{y} = 11\ 11\ 01\ 01\ 10\ 11 \qquad \text{(2 errors)}$$
$$\boldsymbol{y} = 11\ 11\ 10\ 01\ 10\ 11 \qquad \text{(4 errors)}.$$

Führe jeweils die Viterbi-Decodierung durch.

10.3. Betrachte das Standardbeispiel $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$ mit dem Punktierungsschema $111xx1$ (x bedeutet punktieren). Welche Coderate hat der punktierte Code? Empfangen wird die (fehlerfreie) Sendefolge (Hard-Decision)

$$\boldsymbol{y} = 11010011.$$

Führe die Viterbi-Decodierung durch.

10.4. Berechne die Viterbi-Metrik für den asymmetrischen Kanal mit den transition probabilities

$P(y\|x)$	$y=0$	$y=1$
$x=0$	0.9	0.1
$x=1$	0.3	0.7

Durch geeignete Näherungen ist die Metrik in eine möglichst einfache Form zu bringen. Interpretiere das Ergebnis.

10.5. Betrachte das Standardbeispiel $\boldsymbol{G}(x) = (1+x+x^2, 1+x^2)$ mit truncation und der Viterbi-Metrik

$\mu(y\|x)$	$y=0$	$y=0'$	$y=1'$	$y=1$
$x=0$	6	3	1	0
$x=1$	0	1	3	6

Decodiere die Empfangsfolge

$$\boldsymbol{y} \;=\; 10'\ 01\ 1'0'\ 1'1'\ 00\ 1'0'\ 1'1\ 00'$$

mit dem Viterbi-Algorithmus. Vergleiche $\mu_8(1)$ mit der Metrik bei fehlerfreier Übertragung.

10.6. Berechne die Viterbi-Metrik for the AWGN channel with octal quantization aus Bild 1.5 in zwei Formen, so daß die Metrik (a) möglichst einfach mit vielen Nullen in der Tabelle ausfällt bzw. (b) möglichst der Metrik für ideale Soft-Decision nahekommt.

10.7. Betrachte das Standardbeispiel $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$ mit Terminierung und dem oktal quantisierten AWGN channel mit der Viterbi-Metrik

$\mu(y\|x)$	$y=0$	$y=0'$	$y=0''$	$y=0'''$	$y=1'''$	$y=1''$	$y=1'$	$y=1$
$x=0$	8	5	3	1	0	0	0	0
$x=1$	0	0	0	0	1	3	5	8

Zur Infofolge 110100 wird $\boldsymbol{a} = 11\ 01\ 01\ 00\ 10\ 11$ gesendet und

$$\boldsymbol{y} \;=\; 11''\ 1'''1''\ 0'0''\ 1''1'''\ 1'0\ 0''1$$

wird empfangen. Führe die Viterbi-Decodierung durch. Für Hard-Decision Demodulation mit

$$\boldsymbol{y} \;=\; 11\ 11\ 00\ 11\ 10\ 01$$

ist erneut die Viterbi-Decodierung durchzuführen und mit dem Ergebnis bei Soft-Decision zu vergleichen.

10.8. Zeige ergänzend zu Theorem 9.1: Für die Wahrscheinlichkeit $P(\text{dpath})$, daß zu einem festen Zeitpunkt ein Fehlerereignis beginnt, gilt folgende Abschätzung:

$$P(\text{dpath}) \le \sum_{d=d_\text{f}}^{\infty} \underbrace{\sum_{i,j} t(d,i,j)}_{=\,w_d} \gamma^d. \tag{10.7.1}$$

Speziell für den AWGN mit idealer Soft-Decision gilt eine schärfere Abschätzung ($E_c = E_b/n$):

$$P(\text{dpath}) \le \sum_{d=d_\text{f}}^{\infty} w_d Q\left(\sqrt{2d\frac{E_c}{N_0}}\right). \tag{10.7.2}$$