Unterstützende Materialien zur Vorlesung

# Verfahren zur Kanalcodierung – Teil 6

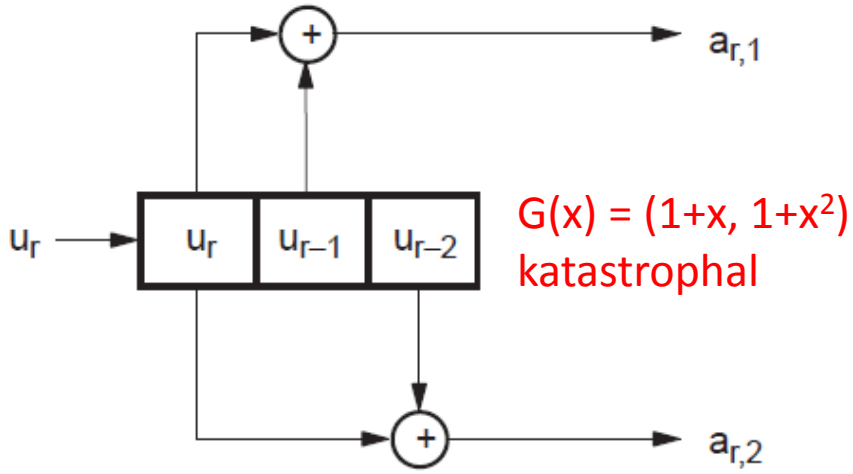
Prof. Dr. Bernd Friedrichs
KIT CEL

Inhalt

- Nicht-katastrophale Encoder und Encoder-Inverses
- Distanzeigenschaften und optimale Faltungscodes
- Trellisdiagramm
- Zustandsdiagramm
- Viterbi-Metrik, Viterbi-Algorithmus für terminierte und nicht-terminierte Codes
- Implementierung und Synchronisation
- Fehlerwahrscheinlichkeit von Faltungscodes, Fehlerstrukturen, Verkette Codes
- Vergleich Blockcodes-Faltungscodes

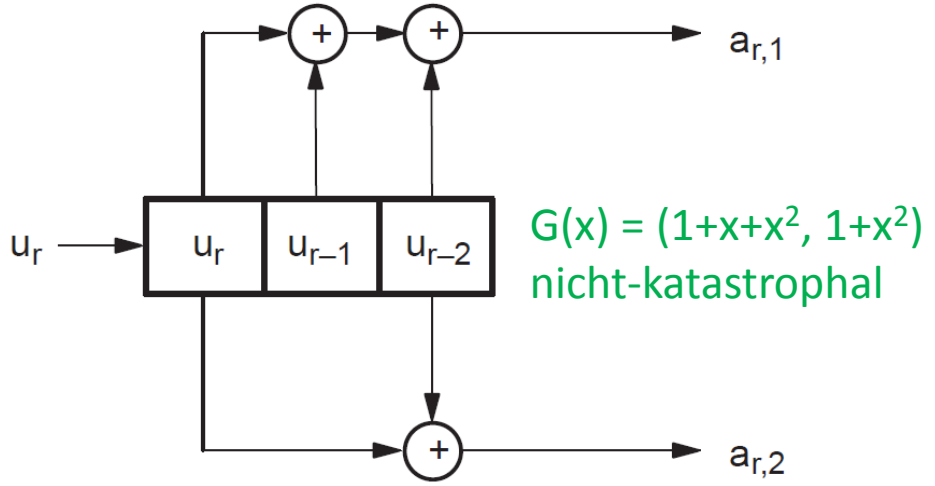
**Beispiel 8.4.** Betrachte den in Bild 8.3 dargestellten Encoder eines $R = 1/2$-Codes mit $\boldsymbol{G}(x) = (1 + x, 1 + x^2)$, der sich nur geringfügig vom Standardbeispiel unterscheidet. Mit Dauer-Eins als Infofolge ergibt sich sofort die Codefolge $\boldsymbol{a}(x) \leftrightarrow (11, 01, 00, 00, 00, \ldots)$. Mit $u(x) = 1 + x + x^2 + x^3 + \cdots = 1/(1 - x)$ folgt das auch aus $u(x)\,\boldsymbol{G}(x) = \dfrac{1}{1-x}\left(1 + x, (1 + x)^2\right) = (1, 1 + x)$.

Beachte: in GF(2) gilt:
$1+x^2 = (1+x)^2$




G(x) = (1+x, 1+x²)
katastrophal

**Bild 8.3.** Katastrophaler $R = 1/2$-Faltungs-Encoder




G(x) = (1+x+x², 1+x²)
nicht-katastrophal

**Bild 8.2.** Standardbeispiel $R = 1/2$-Faltungs-Encoder mit $m = 2$

Mit Dauer-Null und Dauer-Eins gibt es also zwei Infofolgen mit unendlichem Hammingabstand, deren zugehörige Codefolgen sich nur an endlich vielen (nämlich 3) Stellen unterscheiden. Es reichen also 3 Übertragungsfehler aus, um bei der Decodierung unendlich viele Fehler zu produzieren (lawinenartige Fehlerfortpflanzung). Ein solcher Encoder verdient die Bezeichnung katastrophal.

**Satz 8.1.** *Ein nicht-katastrophaler Encoder ist dadurch definiert, daß jede Infofolge unendlichen Gewichts eine Codefolge unendlichen Gewichts impliziert bzw. äquivalent, daß jede endliche Codefolge eine endliche Infofolge impliziert. Das ist äquivalent damit, das die Generatorpolynome keinen gemeinsamen Teiler haben:*

$$\mathrm{GGT}\Big(g_1(x), \ldots, g_n(x)\Big) = 1. \tag{8.4.1}$$

Generell ist der größte gemeinsame Teiler mehrerer Größen als Linearkombination dieser Größen darstellbar (Euklidischer Algorithmus Satz A.8). Es existieren also Polynome $f_1(x), \ldots, f_n(x)$ mit

$$\sum_{\nu=1}^{n} f_\nu(x) g_\nu(x) = \mathrm{GGT}\Big(g_1(x), \ldots, g_n(x)\Big) = 1. \tag{8.4.2}$$

Zur Codefolge $\boldsymbol{a}(x) = \Big(a_1(x), \ldots, a_n(x)\Big) = u(x)\Big(g_1(x), \ldots, g_n(x)\Big)$ gehört also die Infofolge

$$\begin{aligned}\sum_{\nu=1}^{n} a_\nu(x) f_\nu(x) &= \sum_{\nu=1}^{n} u(x) g_\nu(x) \cdot f_\nu(x) \\ &= u(x) \cdot \sum_{\nu=1}^{n} g_\nu(x) f_\nu(x) \;=\; u(x).\end{aligned}$$

Überprüfung des GGT anhand der beiden vorangehenden Beispiele:

- Für den katastrophalen Encoder gilt GGT($1+x$, $1+x^2$) = $1+x$
- Für den nicht-katastrophalen Encoder gilt GGT($1+x+x^2$, $1+x^2$) = 1

Encoder-Inverses versus Decoder:

Die Polynome $f_1(x), \ldots, f_n(x)$ aus (8.4.2) bilden ein Encoder-Inverses, das zur Codefolge die Infofolge liefert, wie am nachfolgenden Beispiel noch genauer erklärt wird. Das Encoder-Inverse ist aber kein Decoder, da es für verfälschte Codefolgen mit Fehlermustern unbrauchbar ist und Soft-Decision überhaupt nicht verarbeiten kann. Im Viterbi-Decoder ist das Encoder-Inverse implizit enthalten, so daß die Polynome $f_1(x), \ldots, f_n(x)$ nicht explizit bekannt sein müssen.

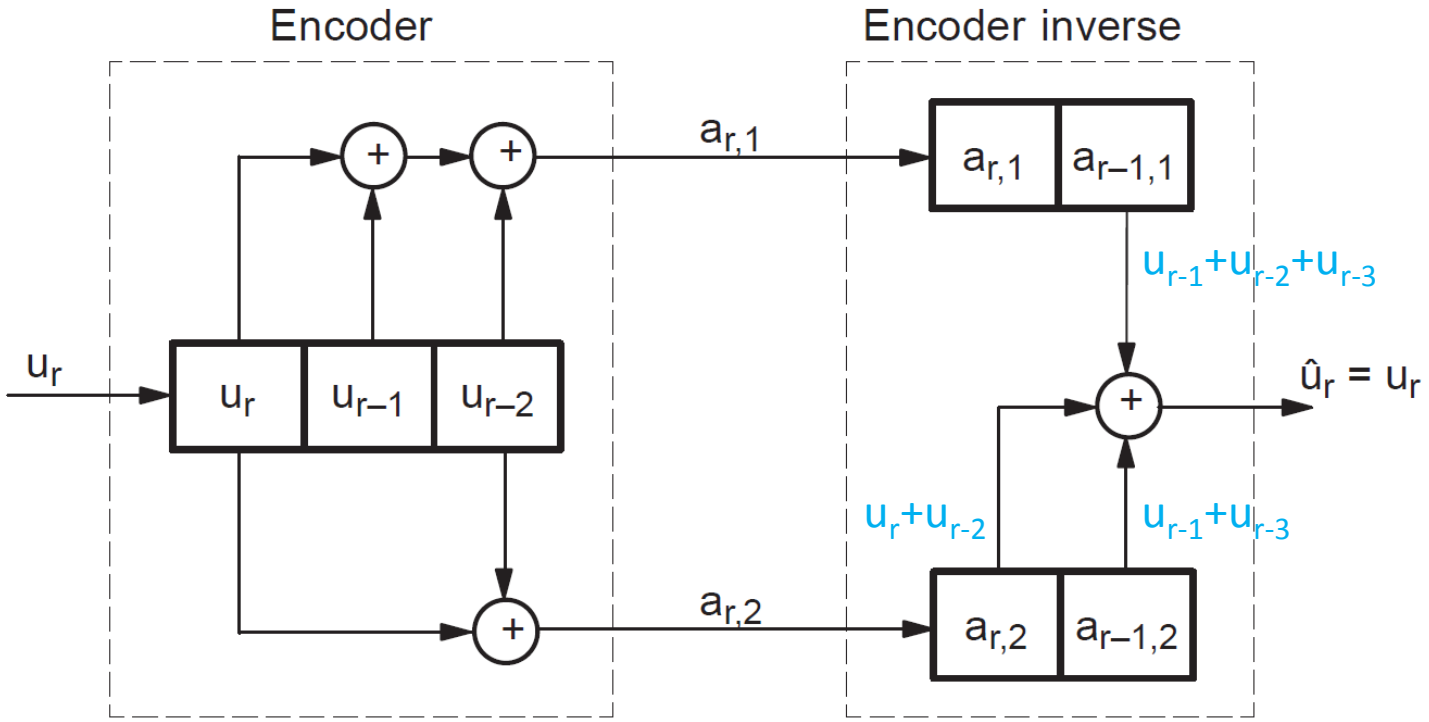



**Bild 8.4.** Standardbeispiel: Encoder – Encoder-Inverses

In der rekursiven Schreibweise gilt:

$$\text{Encoder: } a_{r,1} = u_r + u_{r-1} + u_{r-2} \quad , \quad a_{r,2} = u_r + u_{r-2}$$
$$\text{Encoder-Inverses: } \hat{u}_r = a_{r-1,1} + a_{r,2} + a_{r-1,2}.$$

Das kann über $\hat{u}_r = (u_{r-1} + u_{r-2} + u_{r-3}) + (u_r + u_{r-2}) + (u_{r-1} + u_{r-3}) = u_r$ sofort verifiziert werden. ■

**Beispiel 8.5.** Bei $\boldsymbol{G}(x) = (g_1(x), g_2(x)) = (1+x+x^2, 1+x^2)$ ist $g_1(x)$ irreduzibel und somit liegt beim Standardbeispiel ein nicht-katastrophaler Encoder vor. Mit $(f_1(x), f_2(x)) = (x, 1+x)$ gilt leicht nachprüfbar

$$f_1(x)g_1(x) + f_2(x)g_2(x) = 1.$$

Somit folgt

$$\hat{u}(x) = \boldsymbol{a}(x) \begin{pmatrix} x \\ 1+x \end{pmatrix} = u(x)\big(1+x+x^2, 1+x^2\big) \begin{pmatrix} x \\ 1+x \end{pmatrix} = u(x).$$

**Beispiel 8.6.** Bei $\boldsymbol{G}(x) = (g_1(x), g_2(x)) = (1+x, 1+x^2) = (1+x)(1, 1+x)$ liegt ein katastrophaler Encoder vor. Es existieren keine zwei Polynome $f_1(x), f_2(x)$ mit $f_1(x)g_1(x) + f_2(x)g_2(x) = 1$, denn

$$f_1(x)g_1(x) + f_2(x)g_2(x) = (1+x) \cdot \underbrace{\big(f_1(x) + f_2(x)(1+x)\big)}_{\text{Polynom}} = 1$$

ist nach der Gradformel unmöglich. ■

Die Güte eines Blockcodes wird im wesentlichen mit der Minimaldistanz erfaßt, d.h. mit dem minimalen Hamminggewicht aller Codewörter. Eine entsprechende Charakterisierung für Faltungscodes wird nachfolgend eingeführt:

**Definition 8.2.** *Die freie Distanz $d_\mathrm{f}$ (andere oft verwendete Schreibweisen sind $d_\mathrm{free}$ oder $d_\infty$) eines Faltungscodes ist definiert als das minimale Hamminggewicht aller Codefolgen:*

$$d_\mathrm{f} \;=\; \min\left\{ w_H(u(x)\,\boldsymbol{G}(x)) \;\middle|\; u(x) = \sum_{r=0}^{\infty} u_r x^r \neq 0 \right\}. \qquad (8.5.1)$$

*Mit Ausnahme punktierter Codes kann immer $u_0 = 1$ gesetzt werden. Zwei Codefolgen unterscheiden sich immer an mindestens $d_\mathrm{f}$ Stellen. Ein Faltungscode heißt optimal, wenn $d_\mathrm{f}$ maximal ist gegenüber allen anderen Faltungscodes mit gleicher Coderate und gleicher Gedächtnislänge $m$.*

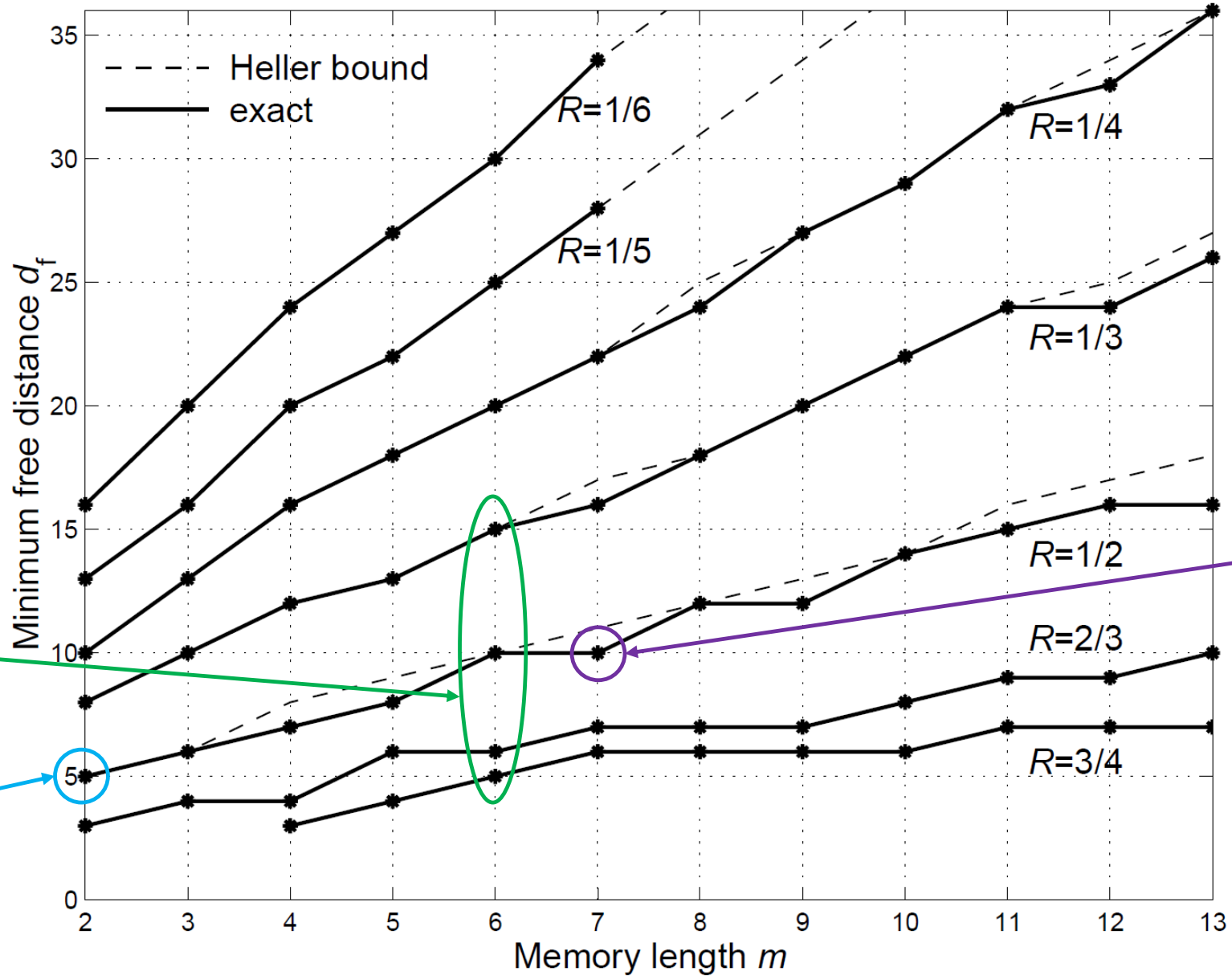
**Table 9.3b.** Generator polynomials in octal notation for $R = 1/2$ and $R = 1/3$ optimum convolutional codes with $m = 2 \ldots 16$ (from [79])

| $m$ | $R = 1/2$ | | | $R = 1/3$ | | | |
|---|---|---|---|---|---|---|---|
| | $g_1(x)$ | $g_2(x)$ | $d_f$ | $g_1(x)$ | $g_2(x)$ | $g_3(x)$ | $d_f$ |
| 2 | 5 | 7 | 5 | 5 | 7 | 7 | 8 |
| 3 | 64 | 74 | 6 | 54 | 64 | 74 | 10 |
| 4 | 46 | 72 | 7 | 52 | 66 | 76 | 12 |
| 5 | 65 | 57 | 8 | 47 | 53 | 75 | 13 |
| 6 | 554 | 744 | 10 | 554 | 624 | 764 | 15 |
| 7 | 712 | 476 | 10 | 452 | 662 | 756 | 16 |
| 8 | 561 | 753 | 12 | 557 | 663 | 711 | 18 |
| 9 | 4734 | 6624 | 12 | 4474 | 5724 | 7154 | 20 |
| 10 | 4672 | 7542 | 14 | 4726 | 5562 | 6372 | 22 |
| 11 | 4335 | 5723 | 15 | 4767 | 5723 | 6265 | 24 |
| 12 | 42554 | 77304 | 16 | 42554 | 43364 | 77304 | 24 |
| 13 | 43572 | 56246 | 16 | 43512 | 73542 | 76266 | 26 |
| 14 | 56721 | 61713 | 18 | | | | |
| 15 | 447254 | 627324 | 19 | | | | |
| 16 | 716502 | 514576 | 20 | | | | |

Standardbeispiel

Industriestandard-Code

Beispiel zur oktalen Notation: $(53, 75)_{\text{octal}} = (1 + x^2 + x^4 + x^5, 1 + x + x^2 + x^3 + x^5)$




Industriestandard-Code, R=1/3, 1/2, 2/3, 3/4, m=6

Standardbeispiel R=1/2, m=2

R=1/2, m=7 wäre unsinnig weil das gegenüber m=6 den Aufwand verdoppelt (siehe später) ohne Zuwachs bei $d_f$

**Figure 9.7.** Minimum free distance versus memory length for various code rates

Zusammenfassung Beispiele:

| | | | |
|---|---|---|---|
| $G(x) = (1+x+x^2, 1+x^2)$, | $d_f=5$, | nicht-katastrophal, | optimal, eindeutig bestimmt |
| $G(x) = (1+x, 1+x^2)$, | $d_f=4$, | katastrophal | nicht verwenden! |
| $G(x) = (1+x+x^2, 1+x+x^2)$, | $d_f=4$ (und nicht 6), | katastrophal | nicht verwenden! |
| $G(x) = (1+x+x^2, 1)$, | $d_f=4$, | nicht-katastrophal, systematisch | nicht verwenden! |

denn $u(x) = 1 + x$ erzeugt $\boldsymbol{a}(x) = (1 + x^3, 1 + x^3)$
(minimales Gewicht der Codefolge tritt hier bei nicht-minimalem Gewicht der Infofolge auf)

Für die Infofolge $u(x) = 1$ ergibt sich die Codefolge $\boldsymbol{a}(x) = \boldsymbol{G}(x)$ und daraus resultiert eine obere Grenze für die freie Distanz:

$$d_\mathrm{f} \le w_H(\boldsymbol{G}(x)) = \sum_{\nu=1}^{n} w_H(g_\nu(x)) \qquad (8.5.2)$$

$$\le n \cdot (m+1). \qquad (8.5.3)$$

Oftmals gilt in der ersten Abschätzung sogar Gleichheit, während die zweite Abschätzung bei nicht-katastrophalen Codes nie scharf sein kann. Bei einem systematischen $R = 1/2$-Code gilt $\boldsymbol{G}(x) = (g_1(x), 1)$ und somit

$$d_\mathrm{f} \le m + 2.$$

Vergleiche:
$d_f=5$ beim Standardbeispiel R=1/2, m=2
(wird noch gezeigt)

Da Tabelle 8.4 für die optimalen Codes größere freie Distanzen zeigt, können optimale Codes also nicht systematisch sein.

Bei terminierten Faltungscodes gilt für die Minimaldistanz des entstehenden Blockcodes immer $d_{\min} = d_{\mathrm{f}}$. Allerdings sind das asymptotisch schlechte Blockcodes im Sinn von Abschnitt 3.4, denn bei konstanter Coderate bleibt die Minimaldistanz konstant und somit konvergiert die Distanzrate bei wachsender Blocklänge gegen Null. Dennoch erweisen sich die Faltungscodes als sehr leistungsfähig. Die Festlegung optimaler Codes in Definition 8.2 erfolgt in Hinblick auf die ML-Decodierung mit dem Viterbi-Algorithmus (siehe Kapitel 9).

## Ausblick

Die Beschreibung von Faltungscodes bzw. ihrer Encoder erfolgte bisher mit Polynomen oder Schieberegistern. Zur Beschreibung der Codemenge selbst wird in Abschnitt 8.6 das Trellisdiagramm (wichtig für die ML-Decodierung) und in Abschnitt 8.7 das Zustandsdiagramm (wichtig für die Berechnung der Distanzeigenschaften) eingeführt.

$$z_r = \zeta_i$$

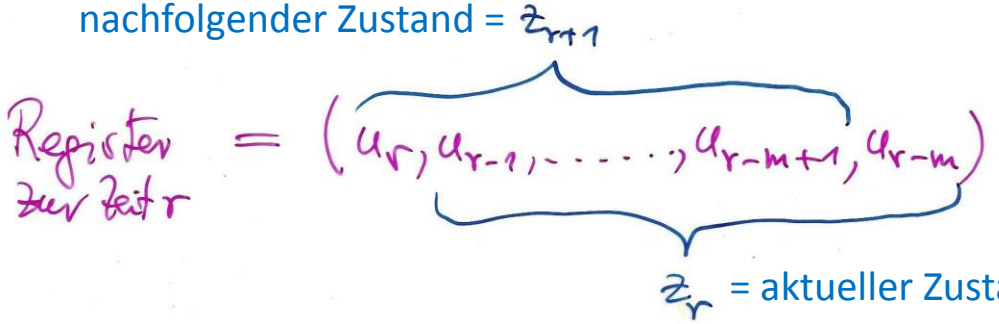
Nach (8.1.1) ist der aktuelle Codeblock $\boldsymbol{a}_r$ eine Funktion des aktuellen Infobits $u_r$ und der $m$ vorangehenden Infobits $u_{r-1}, \ldots, u_{r-m}$. Das binäre $m$-Tupel $(u_{r-1}, \ldots, u_{r-m})$ wird als *Zustand* angesprochen. Die Werte dieser $2^m$ Zustände werden mit $\zeta_1, \ldots, \zeta_{2^m}$ bezeichnet. Dabei sei $\zeta_1 = (0, \ldots, 0)$ der *Nullzustand*. Der Zustand zur Zeit $r$ (bezogen auf die Infobits bzw. auf die Codeblöcke) wird mit $z_r \in \{\zeta_1, \ldots, \zeta_{2^m}\}$ bezeichnet. Aus dem aktuellen Infobit und dem aktuellen Zustand ergeben sich der aktuelle Codeblock und der nächste Zustand:

$$\boldsymbol{a}_r = \text{Encoder}(u_r, z_r) \quad , \quad z_{r+1} = \text{Shiftfunktion}(u_r, z_r). \tag{8.6.1}$$

**Beispiel 8.8.** Für das Standardbeispiel mit $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$ werden die Zustände als $\zeta_1 = 00$, $\zeta_2 = 10$, $\zeta_3 = 01$, $\zeta_4 = 11$ durchnumeriert. Dann gilt:

| Encoder | | |
|---|---|---|
| $\boldsymbol{a}_r$ | $u_r = 0$ | $u_r = 1$ |
| $z_r = 00$ | 00 | 11 |
| $z_r = 10$ | 10 | 01 |
| $z_r = 01$ | 11 | 00 |
| $z_r = 11$ | 01 | 10 |

| Shiftfunktion | | |
|---|---|---|
| $z_{r+1}$ | $u_r = 0$ | $u_r = 1$ |
| $z_r = 00$ | 00 | 10 |
| $z_r = 10$ | 01 | 11 |
| $z_r = 01$ | 00 | 10 |
| $z_r = 11$ | 01 | 11 |



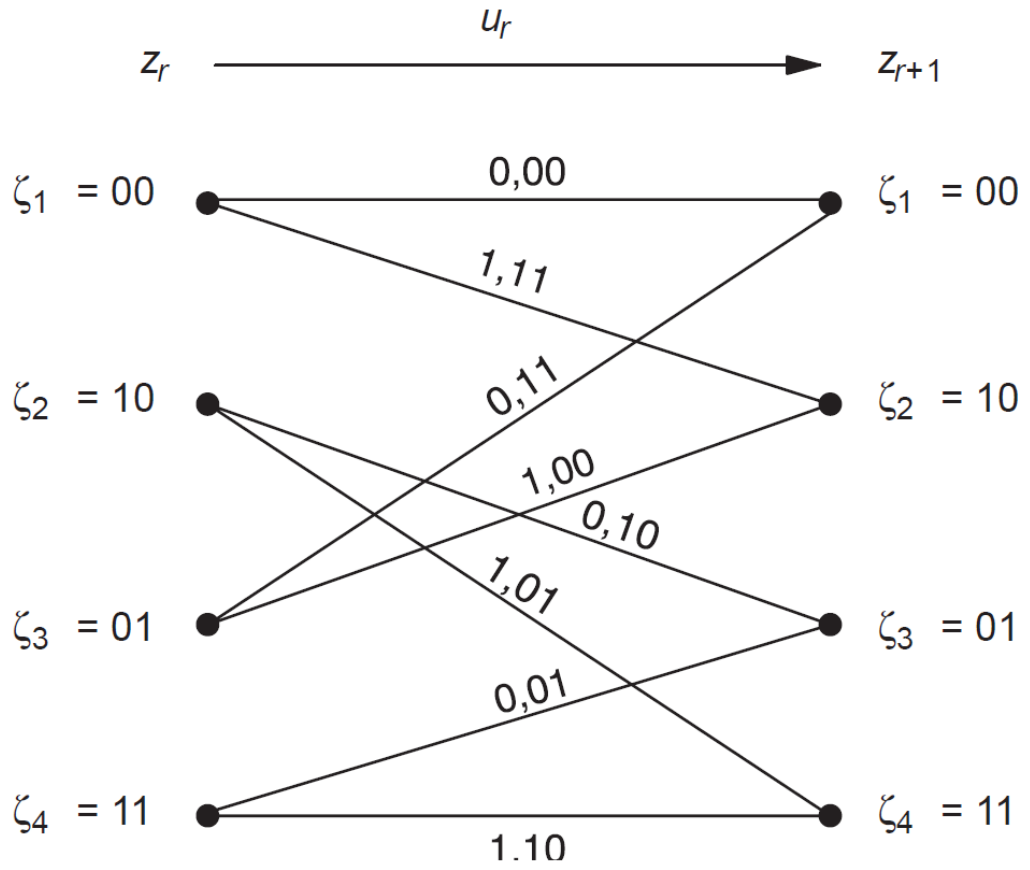
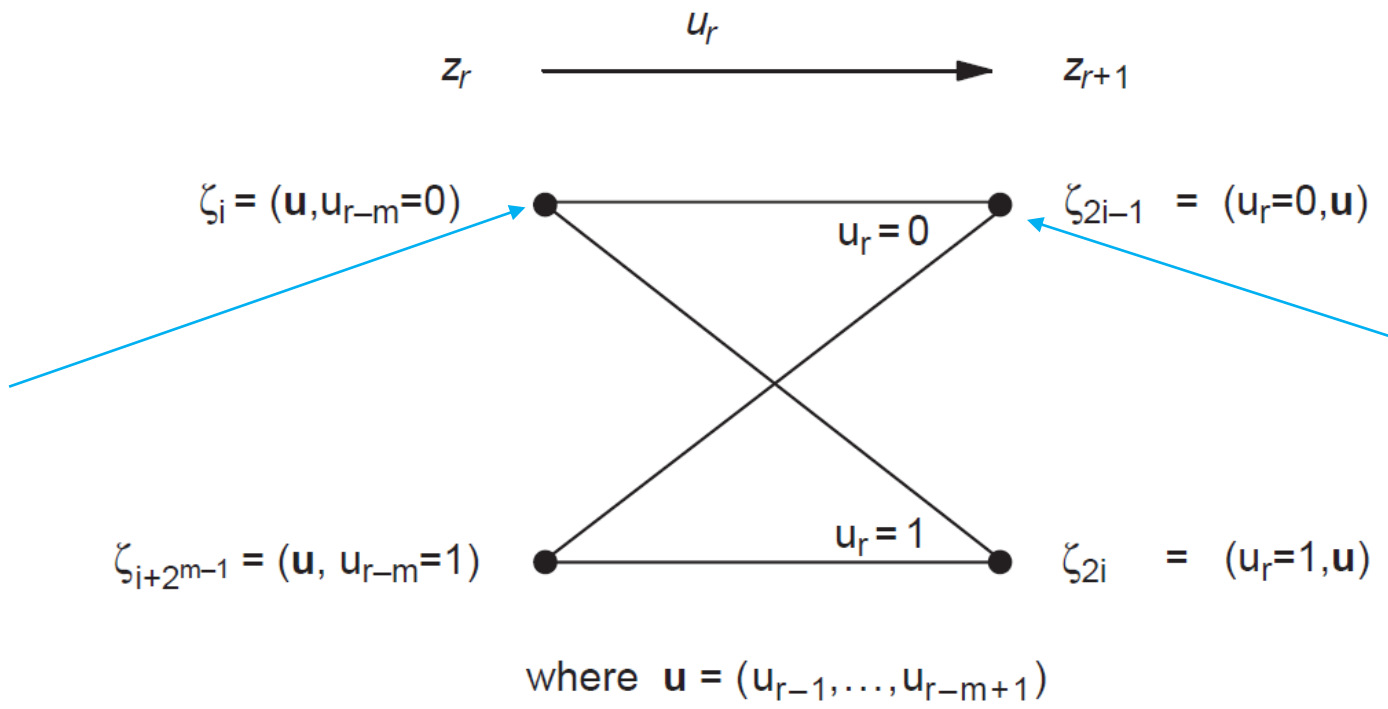



**Bild 8.5.** Trellissegment für das Standardbeispiel $\boldsymbol{G}(x) = (1 + x + x^2, 1 + x^2)$

In Bild 8.5 sind diese Tabellen als Trellissegment dargestellt. Die Kanten zwischen den Zuständen sind mit $u_r$, $\boldsymbol{a}_r$ beschriftet. Die spezielle Generatormatrix wirkt sich nur auf die $\boldsymbol{a}_r$-Beschriftung aus, aber nicht auf die Zustandsübergänge.

Generell hat ein *Trellissegment* $2^m$ Zustände. Bei $R = 1/n$-Codes gehen von jedem Zustand $z_r = \zeta_i$ stets 2 *Kanten* (Zweig, branch) ab und bei jedem Zustand $z_{r+1} = \zeta_j$ kommen stets 2 Kanten an. Wenn die Zustände als umgekehrte Dualzahlen durchnumeriert werden, stellt sich die mit Bild 8.6 formulierte Symmetrie ein.

Von $\zeta_i$ und $\zeta_{i+2^{m-1}}$ gehen insgesamt 4 Kanten ab, die bei $\zeta_{2i-1}$ und $\zeta_{2i}$ ankommen. Die Beschriftung der Kanten mit $u_r$ kann entfallen, da obere abgehende Kanten stets $u_r = 0$ und untere abgehende Kanten stets $u_r = 1$ entsprechen.




Von jedem Zustand gehen 2 Kanten ab

Bei jedem Zustand kommen 2 Kanten an

**Bild 8.6.** Elementarteil des allgemeinen Trellissegmentes

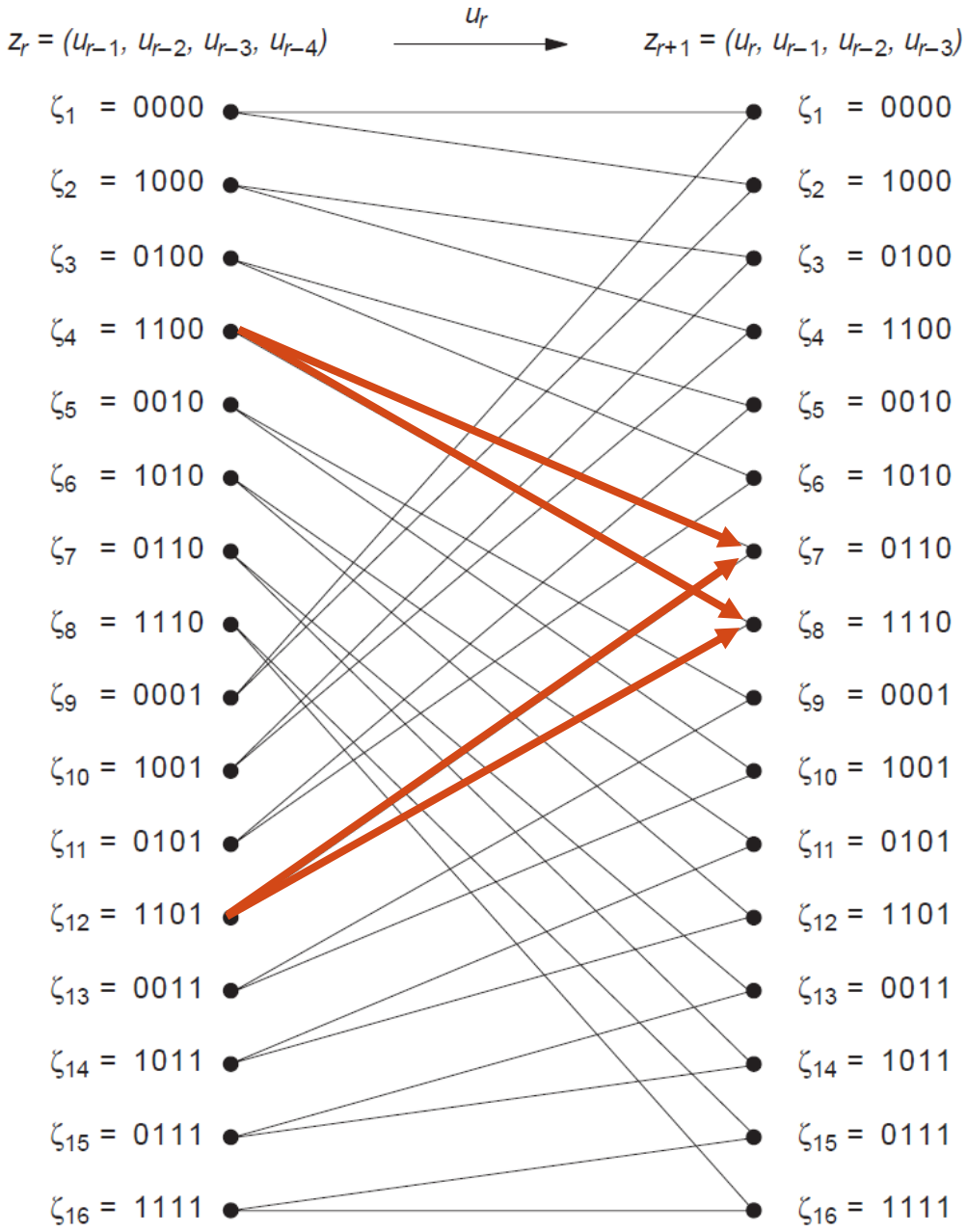



**Bild 8.7.** Trellissegment für Gedächtnislänge $m = 4$

Noch deutlicher werden die Symmetrien anhand eines komplizierteren Beispiels mit m=4.

Kanten von i und $i+2^{m-1}$ nach 2i-1 und 2i

Hier sind keine Kantenbeschriftungen angegeben, also sind auch keine Codeeigenschaften hinterlegt, sondern das ganze Bild repräsentiert im Kern nur die Trivialinformation m=4.

Ein vollständiges *Trellisdiagramm* (oder kurz Trellis) ergibt sich durch Hintereinander-Reihung der einzelnen Trellissegmente. Aufgrund der Vorbelegung des Schieberegisters mit Nullen tritt dabei der Anfangseffekt auf, daß die Kanten nur vom Nullzustand ausgehen können. Bei terminierten Codes gibt es einen entsprechenden Endeffekt, da die Kanten wieder auf den Nullzustand zurückgehen müssen. Der Trellis ist ein gerichteter Graph mit einem Anfangs- und einem Endzustand. Anstelle von Zuständen spricht man auch von *Knoten* (node). Die ersten $m$ und die letzten $m$ Segmente sind unvollständig. Eine Kantenfolge wird als *Weg* bezeichnet. Bei nicht-terminierten Codes können die Wege auch unendliche Länge haben.

**Beispiel 8.9.** Das Trellisdiagramm für das Standardbeispiel zeigt Bild 8.8, wobei nach 4 Infobits terminiert wird. Die Kanten sind mit den Codeblöcken beschriftet. Dem dick markierten Weg entsprechen

| | |
|---|---|
| Infofolge | $1, 1, 0, 1, (0, 0)$ |
| Codefolge | $11, 01, 01, 00, 10, 11$ |
| Zustandsfolge | $\zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_2, \zeta_3, \zeta_1$ |

Die gleiche Codefolge wurde in Beispiel 8.3 bereits über G(x) berechnet.

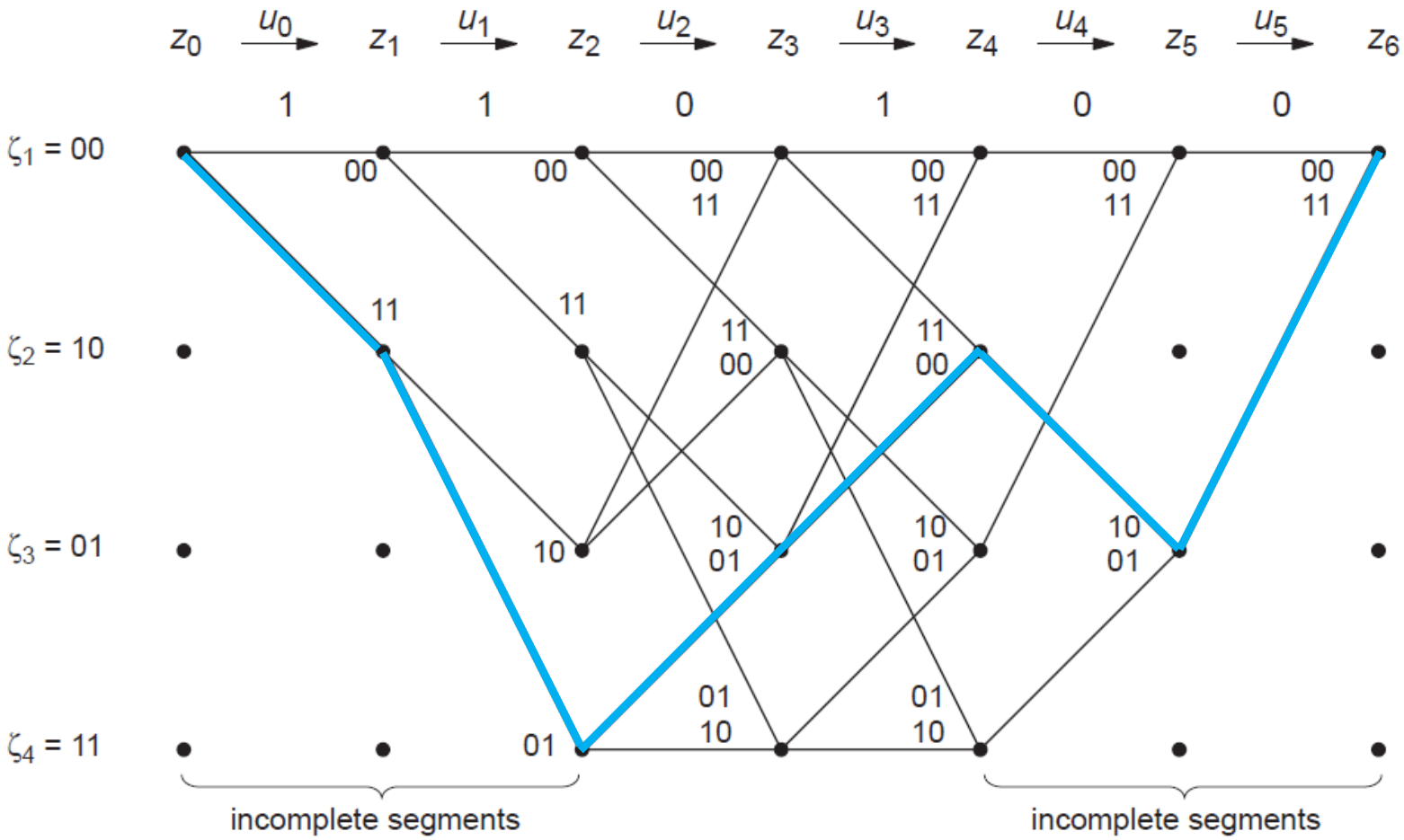



**Bild 8.8.** Trellisdiagramm für das Standardbeispiel

Infofolgen und Codefolgen und Zustandsfolgen entsprechen einander jeweils umkehrbar eindeutig. Mit dem Konzept "Wege durch den Trellis" ist die Codemenge erheblich besser zu überblicken als mit der Generatormatrix. Als *Fundamentalweg* wird ein endlich langer Weg durch den Trellis bezeichnet, der irgendwo beim Nullzustand beginnt und beim Nullzustand endet und zwischendurch den Nullzustand nicht berührt. Die Länge des Fundamentalweges beträgt mindestens $m + 1$ Segmente bzw. $(m + 1)n$ Codebits. Jeder Weg im terminierten Trellis zerfällt in eine endliche Abfolge von Fundamentalwegen und Nullstücken.

Die freie Distanz $d_f$ ergibt sich aus dem Trellis als minimales Hamminggewicht aller beim Nullzustand beginnenden Wege. Das Hamminggewicht bezieht sich dabei natürlich auf die Beschriftung der Kanten mit den Codeblöcken. Ein terminierter Code und ein nicht-terminierter Code haben die gleiche freie Distanz.

**Satz 8.2.** *Bei nicht-katastrophalen Codes kann die freie Distanz allein aus den Fundamentalwegen bestimmt werden.*

Hier werden also die Begriff nicht-katastrophal, freie Distanz und Fundamentalweg verknüpft.

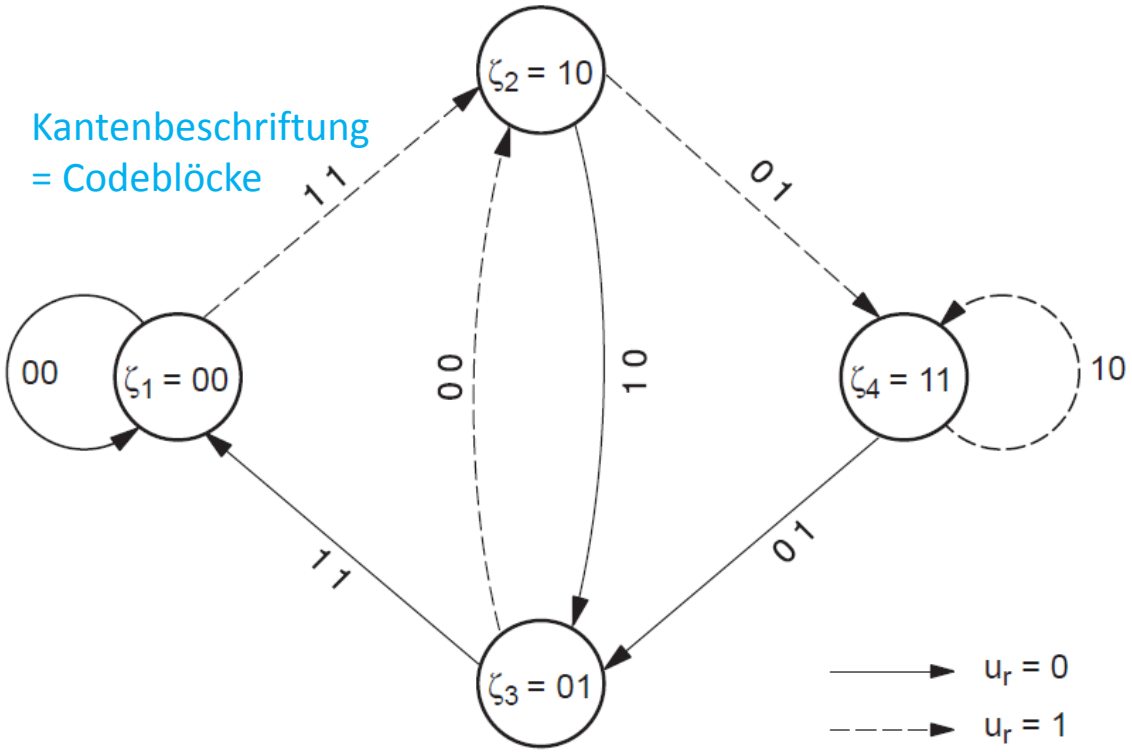



**Bild 8.9.** Zustandsdiagramm für das Standardbeispiel

Beim Trellisdiagramm werden sämtliche Codeeigenschaften schon mit einem einzigen Segment erfaßt. Das *Zustandsdiagramm* (state transition diagram, signal flowchart) ist ein gewichteter und gerichteter Graph, der aus einem Trellissegment dadurch entsteht, daß die Knoten zur Zeit $r+1$ mit denen zur Zeit $r$ identifiziert werden.

Das Prinzip wird sofort klar aus dem in Bild 8.9 dargestellten Zustandsdiagramm für das Standardbeispiel. Eine durchgezogene Kante entspricht $u_r = 0$ und eine gestrichelte Kante entspricht $u_r = 1$. Die Kanten sind mit den Codeblöcken beschriftet.

Allgemein gibt es $2^m$ Zustände mit $2^{m+1}$ Kanten (bei $R = 1/n$). Die möglichen Codefolgen entsprechen den möglichen Zustandssequenzen im Zustandsdiagramm. Das Zustandsdiagramm ist ein endlicher deterministischer Automat (Mealy-Automat). Generell gibt es beim Nullzustand immer eine Schleife mit $u_r = 0$ und $\boldsymbol{a}_r = \boldsymbol{0}$. Trivialerweise ist das Diagramm stets zusammenhängend, da jeder Zustand aus jedem anderen Zustand immer durch eine geeignete Infofolge erreicht werden kann. Die Generatormatrix wirkt sich nur auf die Kantenbeschriftung aus.

Die Fundamentalwege sind im Zustandsdiagramm sehr einfach zu übersehen und darin liegt die Bedeutung dieser Darstellung: Ein Fundamentalweg beginnt im Nullzustand, verläßt den Nullzustand und ist mit dem Wiedererreichen des Nullzustandes beendet. Das minimale Codefolgen-Gewicht aller Fundamentalwege entspricht nach Satz 8.2 der freien Distanz, sofern der Encoder nichtkatastrophal ist.

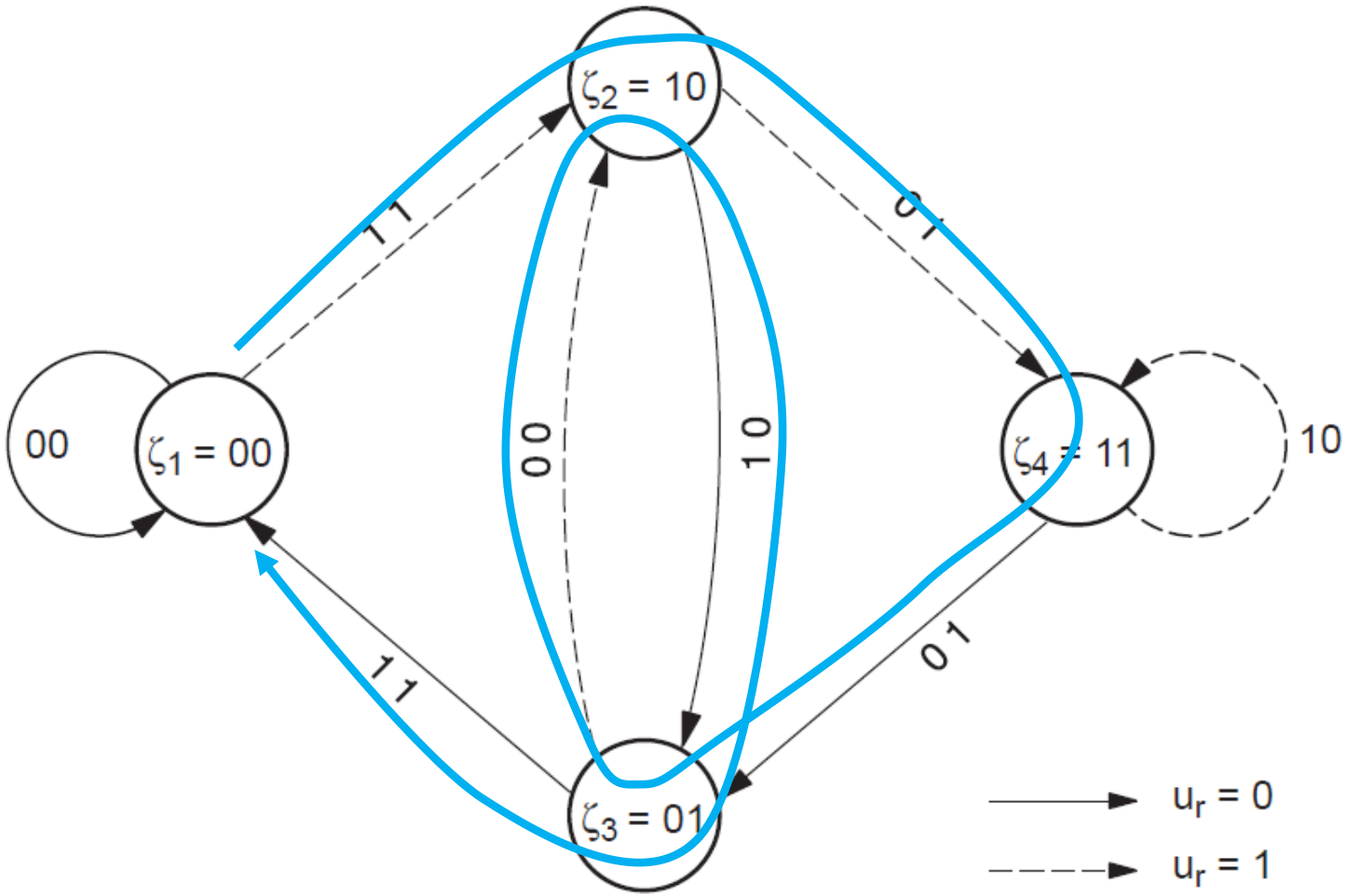



**Bild 8.9.** Zustandsdiagramm für das Standardbeispiel

| | |
|---|---|
| Infofolge | $1, 1, 0, 1, (0, 0)$ |
| Codefolge | $11, 01, 01, 00, 10, 11$ |
| Zustandsfolge | $\zeta_1, \zeta_2, \zeta_4, \zeta_3, \zeta_2, \zeta_3, \zeta_1$ |

Die gleiche Codefolge wurde bereits
- in Beispiel 8.3 über G(x)
- in Beispiel 8.9 über das Trellisdiagramm berechnet

Offensichtlich gibt es keinen Fundamentalweg mit einem Codefolgen-Gewicht kleiner als 5, also $d_f$=5

Die formalisierte Berechnung von $d_f$ erfolgt über die Gewichtsfunktion T(D,I,J), siehe Buch.

Unterscheide zwischen Codefolgen-Gewicht und Infofolgen-Gewicht!

Für die Herleitung der ML-Decodierung kann gedanklich ein terminierter Faltungscode vorausgesetzt werden, der dann ein spezieller Blockcode ist. Nach Satz 1.2 wird zur Empfangsfolge $\boldsymbol{y}$ als ML-Schätzung $\hat{\boldsymbol{a}}$ diejenige Codefolge gewählt, bei der die Übergangswahrscheinlichkeit maximal wird. Für alle Codefolgen $\boldsymbol{b}$ muß also gelten:

$$P_{y|x}(\boldsymbol{y}|\hat{\boldsymbol{a}}) \geq P_{y|x}(\boldsymbol{y}|\boldsymbol{b}). \tag{9.1.1}$$

Für den DMC zerfällt die Übergangswahrscheinlichkeit für Folgen nach Definition 1.1 in ein Produkt von einzelnen Übergangswahrscheinlichkeiten:

$$P_{y|x}(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=0}^{l} P_{y|x}(y_i|x_i). \tag{9.1.2}$$

Dabei ist $l = (L+m)n-1$, wenn nach $L$ Infobits terminiert wird, und $i$ durchläuft die Empfangswerte bzw. Codebits. Anstelle von $P_{y|x}(\boldsymbol{y}|\boldsymbol{x})$ kann auch der skalierte Logarithmus maximiert werden ($\alpha > 0$, Rechnung in den reellen Zahlen):

$$\begin{aligned}
\mu(\boldsymbol{y}|\boldsymbol{x}) &= \alpha \cdot \log P_{y|x}(\boldsymbol{y}|\boldsymbol{x}) + \beta \qquad (9.1.3)\\
&= \alpha \cdot \sum_{i=0}^{l} \log P_{y|x}(y_i|x_i) + \beta \\
&= \sum_{i=0}^{l} \Big(\alpha \cdot \log P_{y|x}(y_i|x_i) + \beta_i\Big) \\
&= \sum_{i=0}^{l} \underbrace{\mu(y_i|x_i)}_{\text{Metrik-Inkremente}}, \qquad (9.1.4)
\end{aligned}$$

Logarithmieren überführt Produkt in Summe, ⇒ ermöglicht damit die Inkrementalisierung.

Maximierung von P(y|x) oder log P(y|x) ist äquivalent.

Durch geschickte Wahl von $\alpha$ und $\beta_i$ sind erstaunliche Formen der Inkremente erreichbar:

Viterbi-Metrik des DMC =

- ist unabhängig vom Code
- MLD bedeutet Maximierung der Metrik

→ Max (durch Wahl von $x_i$ zu Empfangswerten $y_i$)

Inkrementalisierung (additive Zerlegung)

**Beispiel 9.1.** 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:

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

→ Max

Wegen

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

→ Min

ist die Maximierung der Viterbi-Metrik äquivalent mit der Minimierung des Hammingabstandes, wie es in Satz 1.3 für die ML-Decodierung abgeleitet wurde.

**Beispiel 9.2.** Für den AWGN 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

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

sofern $\beta$ 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:

$$\mu(\boldsymbol{y}|\hat{\boldsymbol{a}}) = \sum_{i=0}^{l} y_i \hat{a}_i = \sum_{i=0}^{l} \mu(y_i|\hat{a}_i). \qquad (9.1.5)$$

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

Die Inkrementalisierung erfolgt jetzt nicht Bit-weise sondern zweckmäßigerweise auf Basis der Codeblöcke, also jeweils n Codebits bzw. Empfangswerte bilden das Inkrement.

Zur Vereinfachung wird für die nachfolgenden Beispiele (und nur dafür!)

- Hard-Decision angenommen und
- es wird nicht die obige Metrik maximiert oder der Hammingabstand minimiert, sondern die Anzahl der Übereinstimmungen maximiert (nur zur Schreibvereinfachung - inhaltlich sind alle Methoden identisch)

In der Praxis wird jedoch immer mit der Metrik gearbeitet, weil das auch die problemlose Einbeziehung von Punktierung erlaubt (siehe später).

In diesem Abschnitt wird der Faltungscode als terminiert und im nächsten Abschnitt als nicht-terminiert vorausgesetzt. Alle Kanten werden mit dem Metrik-Inkrement für einen Block beschriftet, d.h. die Inkrementalisierung erfolgt jetzt zweckmäßigerweise mit den Codeblöcken anstelle der Codebits:

$$\mu(\boldsymbol{y}_r|\boldsymbol{a}_r) = \sum_{\nu=1}^{n} y_{r,\nu} a_{r,\nu}. \tag{9.2.1}$$

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.

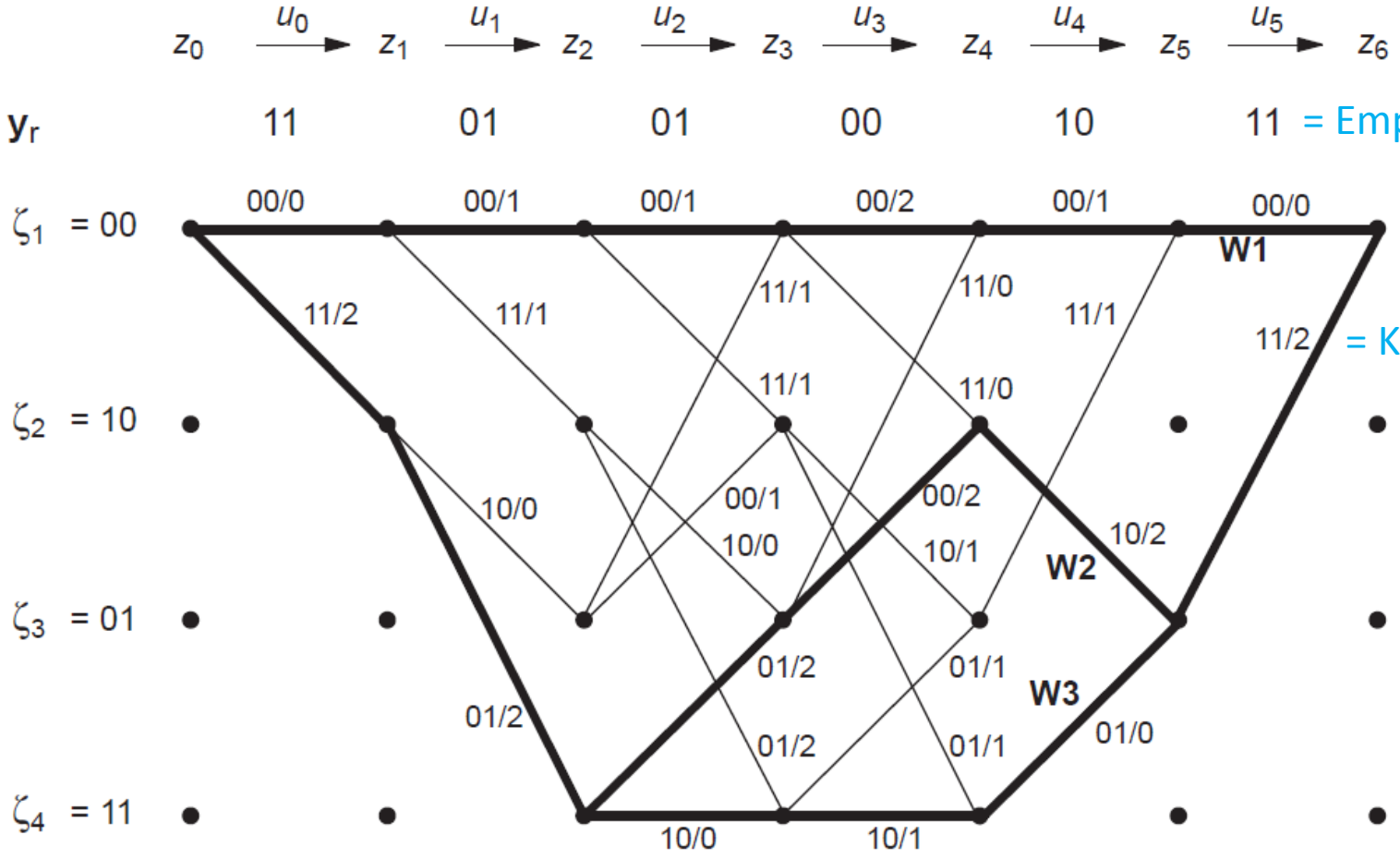



**Bild 9.1.** Standardbeispiel mit ungestört empfangener Codefolge

= Empfangsfolge

= Kantenbeschriftung = Codeblock / „Anzahl Übereinstimmungen zwischen Empfangsfolge und Codeblock“

Für die drei dick markierten Wege gilt
$\mu(y/W_1) = 5$
$\mu(y/W_2) = 12$
$\mu(y/W_3) = 7$
Also entspricht $W_2$ der ungestört empfangenen Codefolge

**Beispiel 9.3.** In Bild 9.1 wird wie in Bild 8.8 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.

Mit jeder Verlängerung des Trellis um ein Segment bzw. Codeblock bzw. Infobit verdoppelt sich die Anzahl der zu untersuchenden Wege. Der Decodieraufwand wächst also exponentiell mit der Länge der Infobitfolge. Der *Viterbi-Algorithmus* (VA) reduziert nun den Aufwand auf ein lineares Wachstum – und zwar ohne den geringsten Verlust an Codierungsgewinn.

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

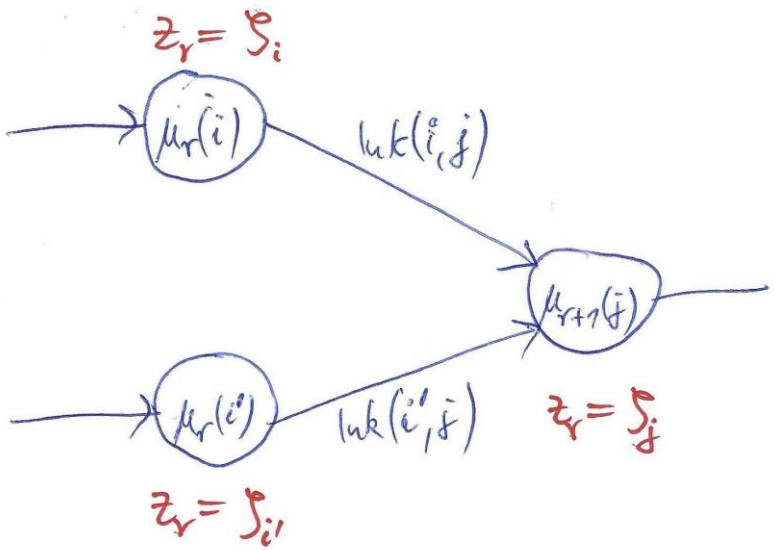
$$\mu_r(i) = \text{Metrik des Weges von } z_0 = \zeta_1 \text{ nach } z_r = \zeta_i \tag{9.2.2}$$

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

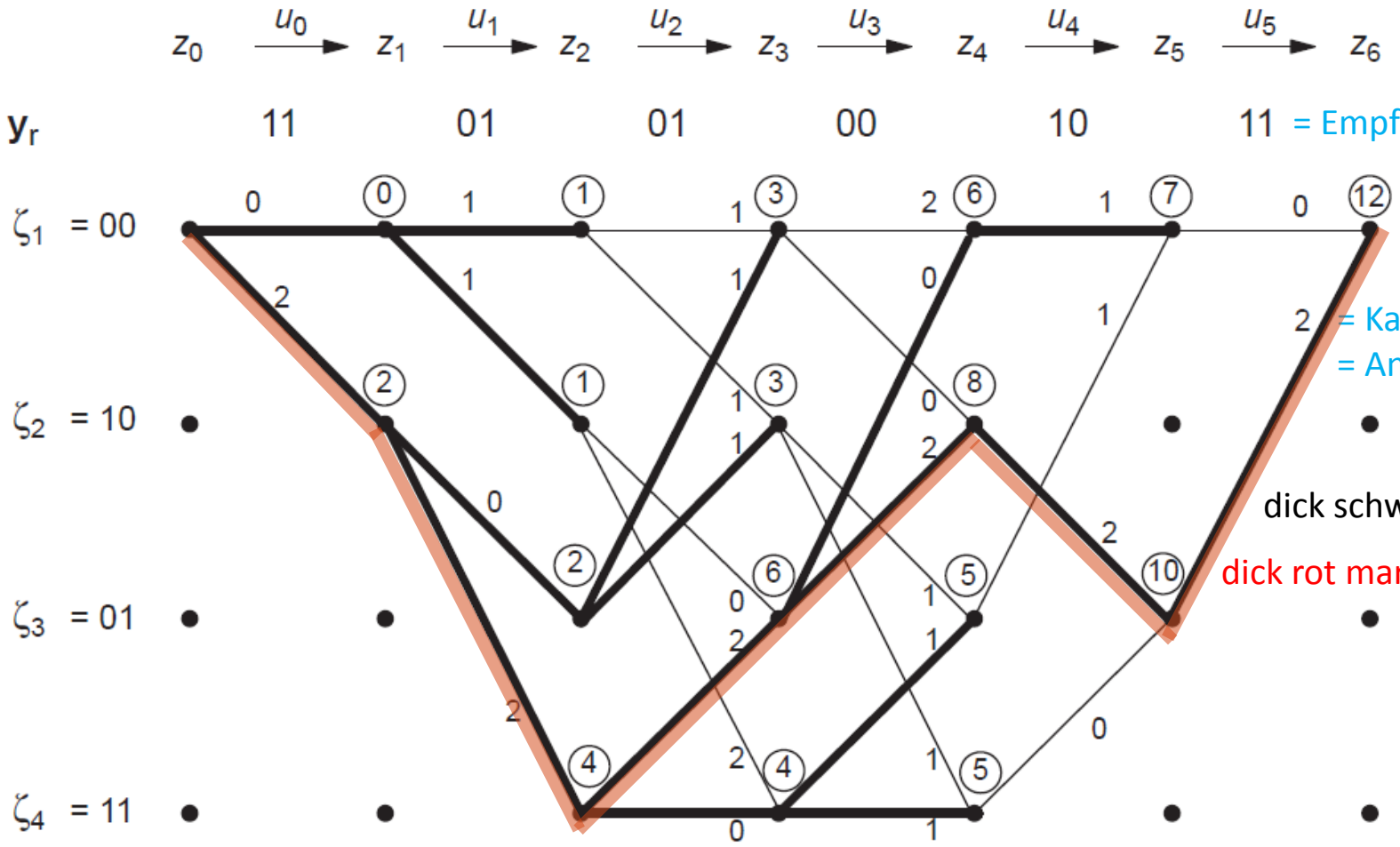
$$\mu_{r+1}(j) = \max \left\{ \begin{array}{l} \mu_r(i) + \text{ Inkrement zur Kante } (\zeta_i, \zeta_j), \\ \mu_r(i') + \text{ Inkrement zur Kante } (\zeta_{i'}, \zeta_j) \end{array} \right\} \tag{9.2.3}$$

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.

Grundidee: Von 2 an einem Zustand ankommenden Wegen überlebt nur einer




$$\mu_{r+1}(j) = \max \left\{ \begin{array}{l} \mu_r(i) + \text{Ink}(i,j), \\ \mu_r(i') + \text{Ink}(i',j) \end{array} \right\}$$




= Survivor-Metrik $\mu_r(i)$ = Summe Inkremente

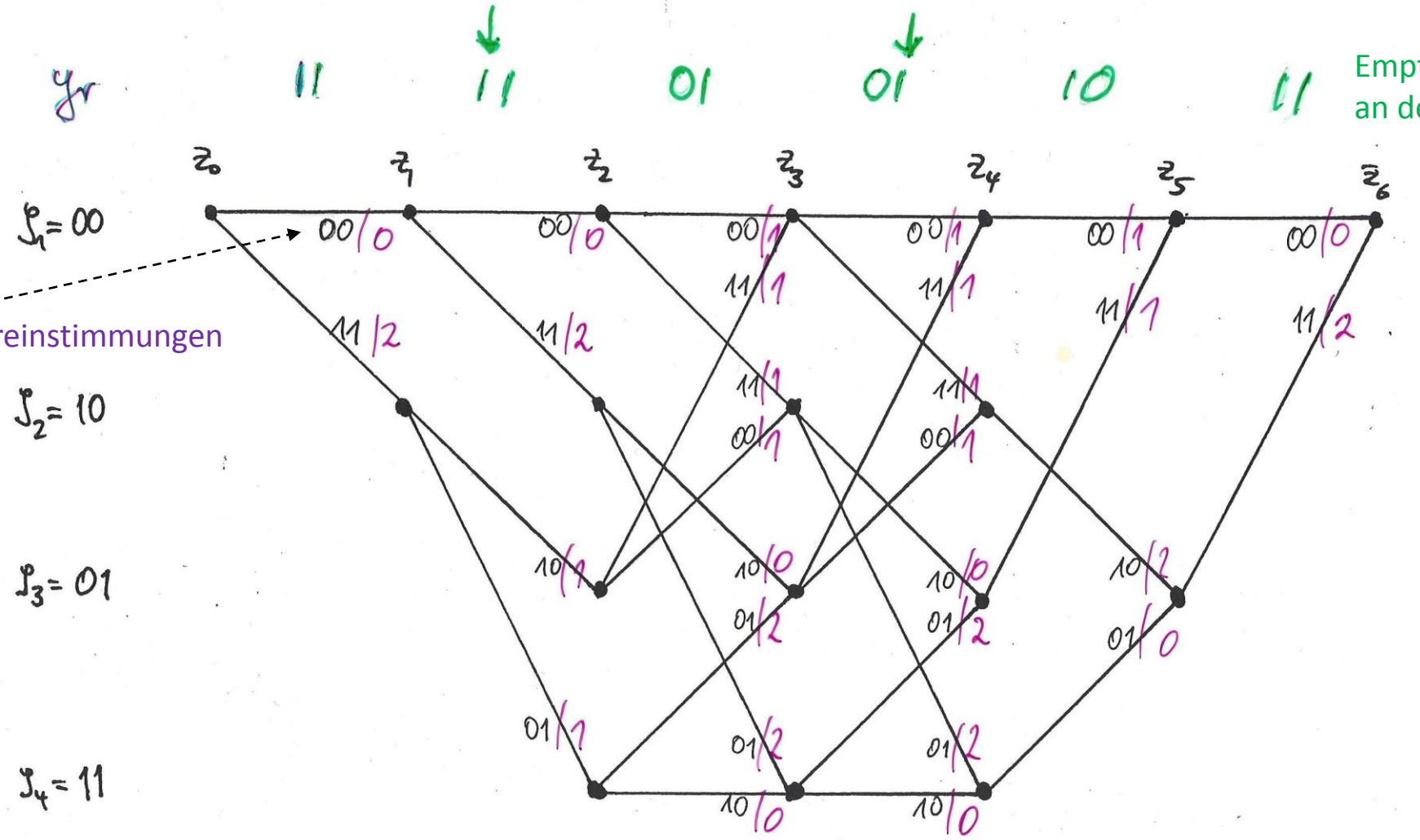
= Kantenbeschriftung = Inkrement
= Anzahl Übereinstimmungen zwischen y und a

dick schwarz markiert = Survivor-Wege an jedem Knoten

dick rot markiert = Survivor-Weg rückwärts vom Endzustand
(kommt automatisch am Anfangszustand an)

**Bild 9.2.** Standardbeispiel mit Terminierung: VA für ungestörte Codefolge

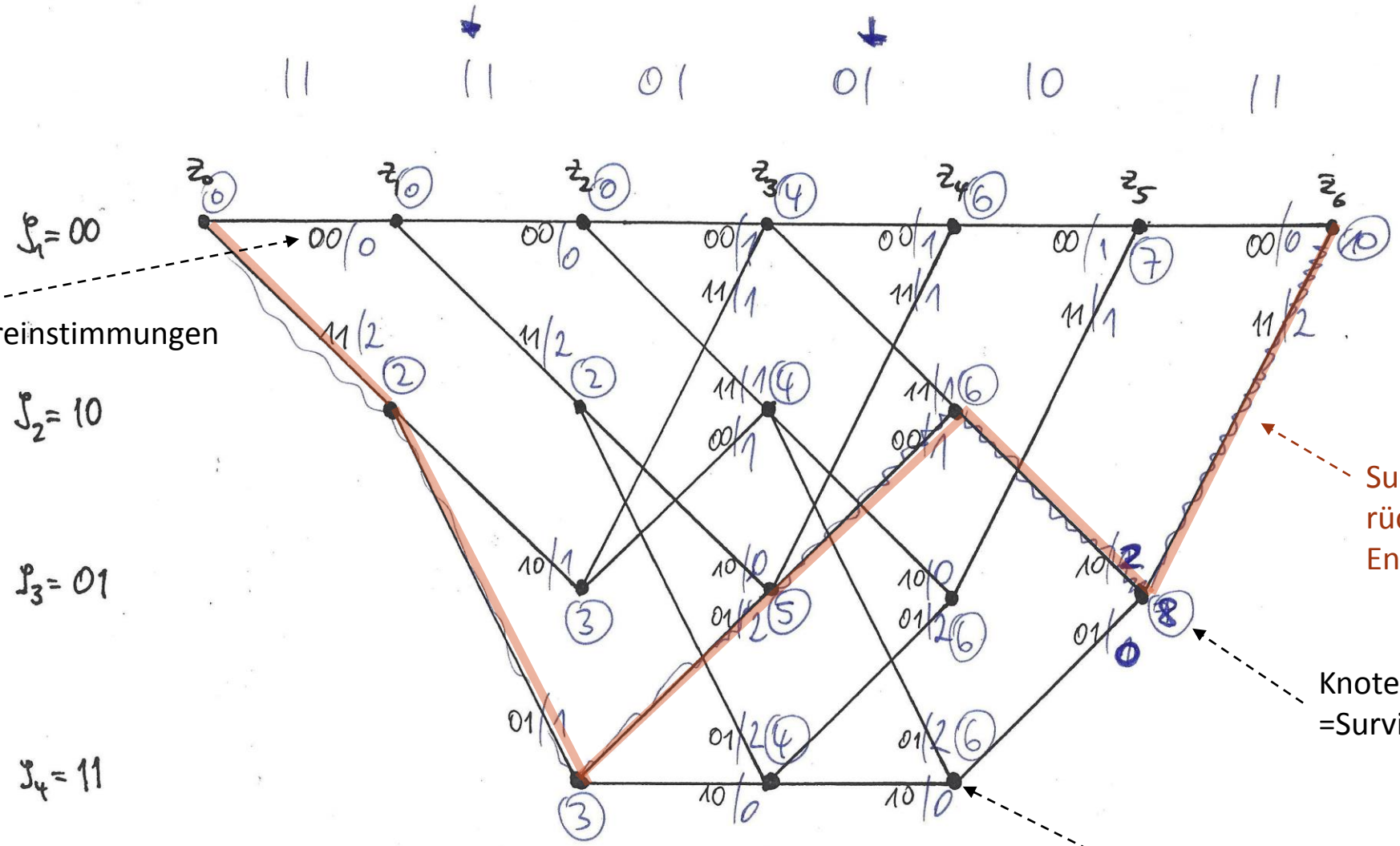
Survivor: untere untere obere untere (obere) (obere) Kante, $\Rightarrow$ Schätzung = 1 1 0 1 (0 0)




Empfangsfolge mit 2 Fehlern an den markierten Stellen

Kantenbeschriftung = Codeblock / #Übereinstimmungen

$G(x) = (1 + x + x^2, 1 + x^2)$ Beispiel S.6 2 Fehler



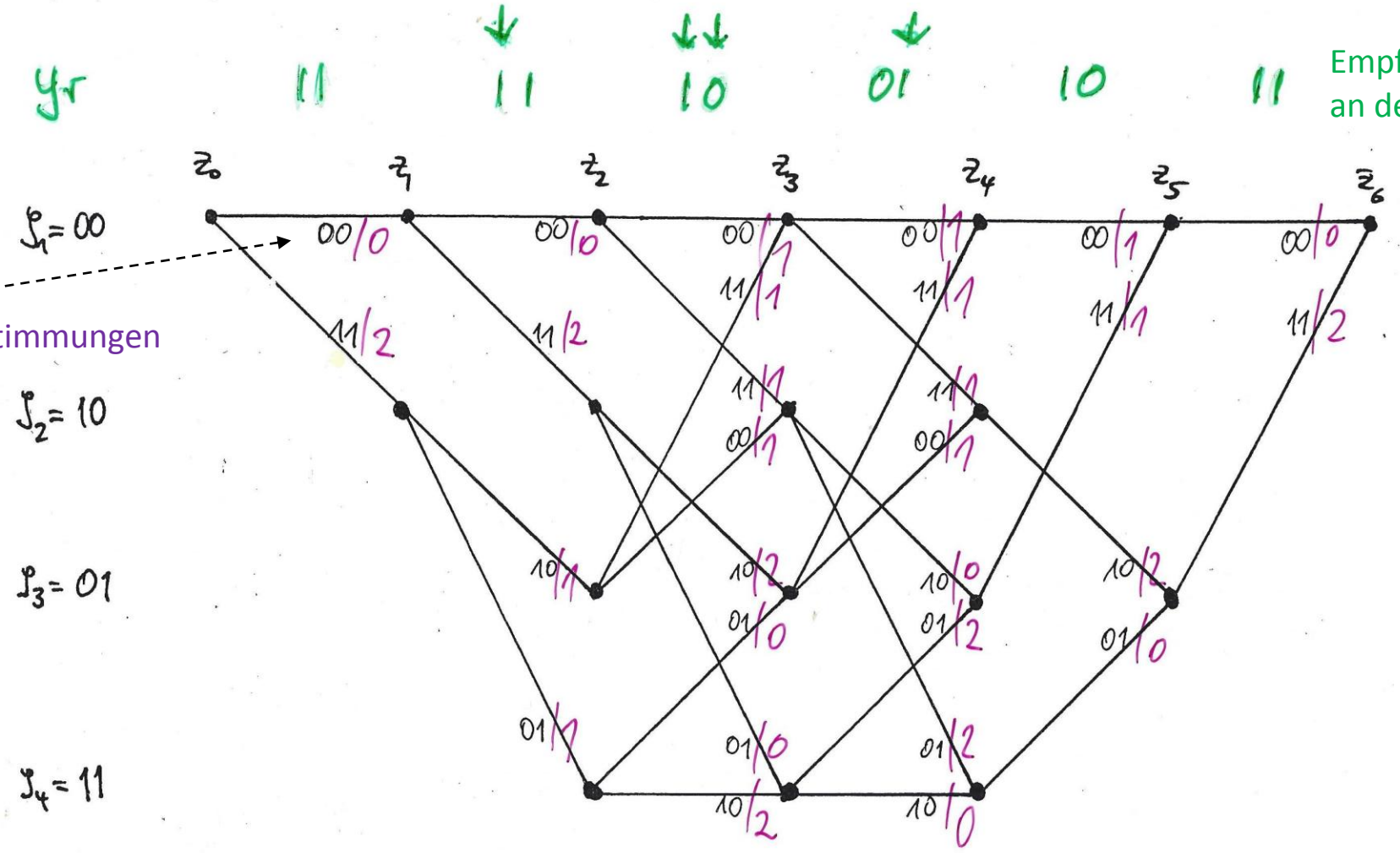

Kantenbeschriftung
= Codeblock / #Übereinstimmungen

Survivor-Weg
rückwärts vom
Endzustand

Knotenbeschriftung
=Survivor-Metrik $\mu_r(i)$

Die an einem Zustand ankommenden
Survivor-Kanten sind nicht hervorgehoben

$G(x) = (1+x+x^2, 1+x^2)$ Beispiel 9.6 2 Fehler

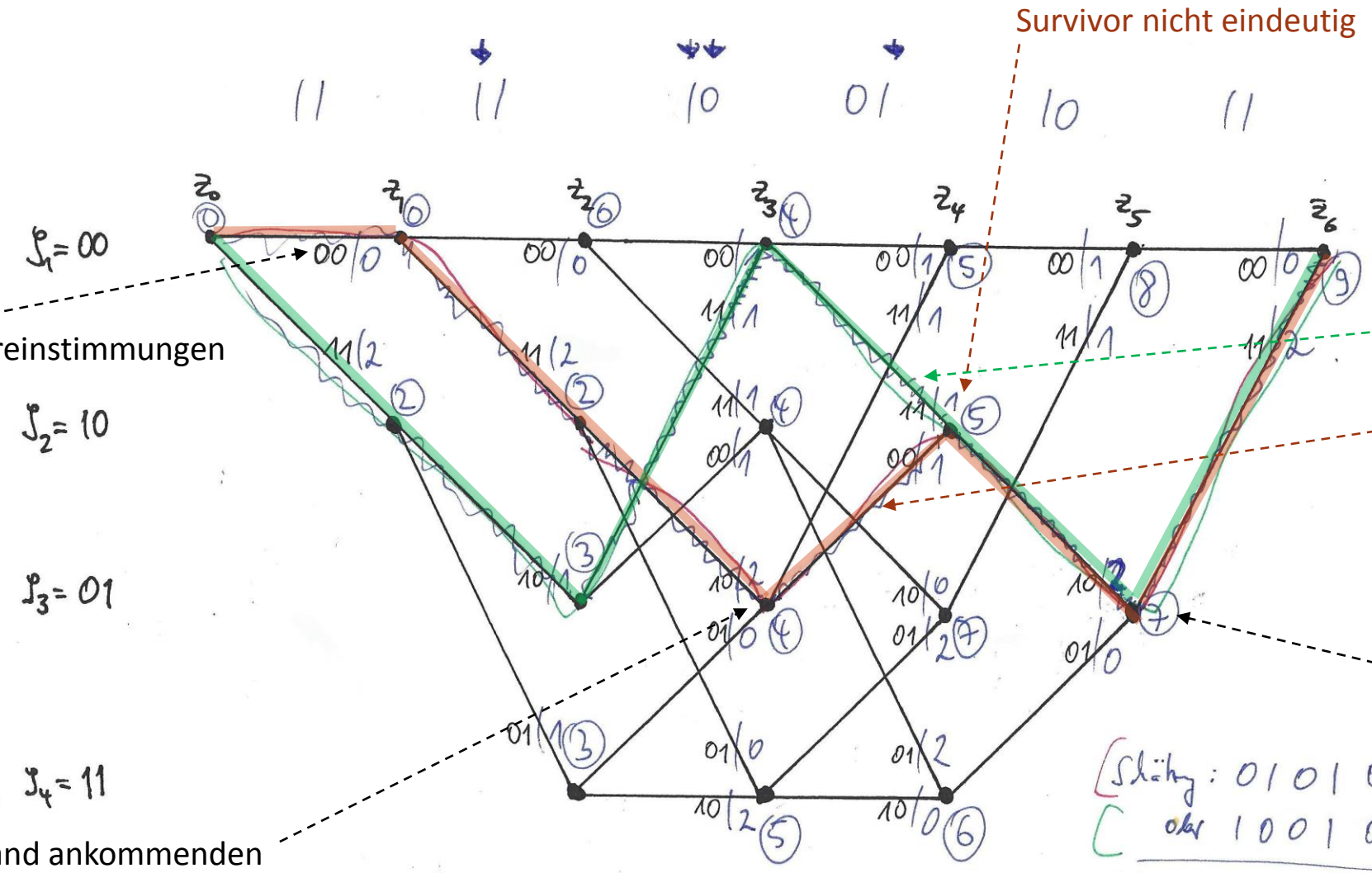



Empfangsfolge mit 4 Fehlern an den markierten Stellen

Kantenbeschriftung = Codeblock / #Übereinstimmungen

$G(x) = (1 + x + x^2, 1 + x^2)$

Beispiel 9.6: 4 Fehler




Kantenbeschriftung
= Codeblock / #Übereinstimmungen

Survivor-Weg

Alternativer
Survivor-Weg
wegen Mehrdeutigkeit

Knotenbeschriftung
= Survivor-Metrik $\mu_r(i)$

Die an einem Zustand ankommenden
Survivor-Kanten sind nicht hervorgehoben

Schätz: 0 1 0 | 0 0 Schätzung rot: falsch
oder 1 0 0 | 0 0 Schätzung grün: falsch
tab. 1 1 0 | 0 0 gesendet

$G(x) = (1+x+x^2, 1+x^2)$ Beispiel 9.6 4 Fehler

Fazit: Beide Schätzungen sind falsch (nicht überraschend da 9 Übereinstimmungen bei 4 Fehlern in 12 Empfangswerten ermittelt wurden)
Also ist hier die Korrekturfähigkeit des Codes überschritten!

Die für Blockcodes sinnvolle Fragestellung "wieviel Fehler sind durch den ML-Decoder korrigierbar" ist bei Faltungscodes 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 Codierungsgewinns.

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 Empfangsfolge werden 5 Fehler angenommen:

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

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.

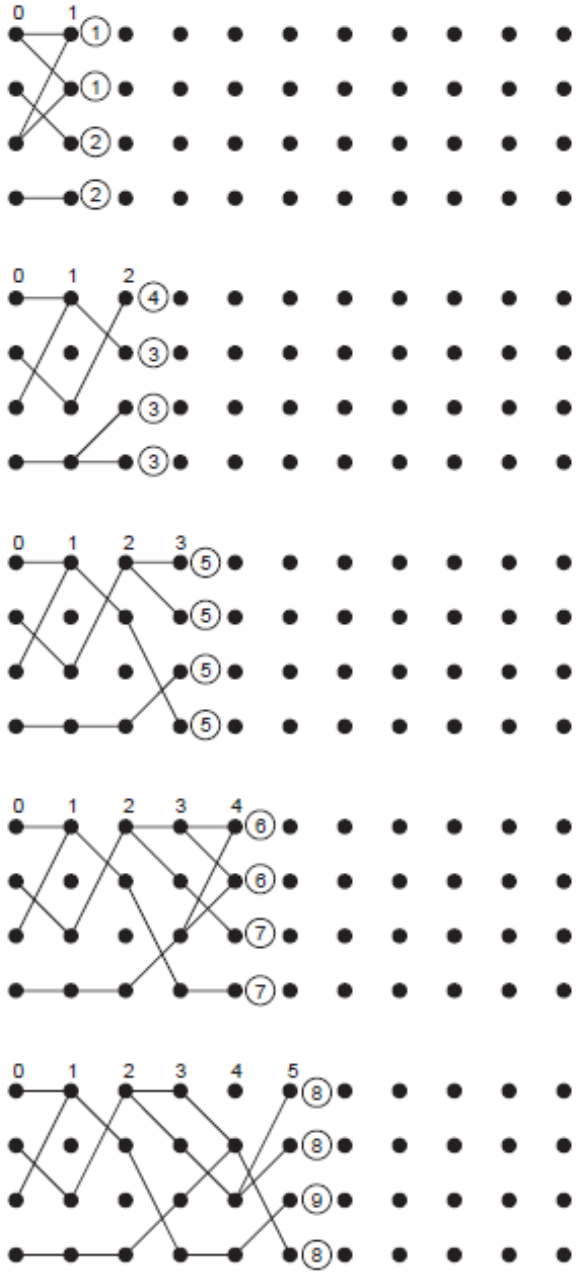


**Bild 9.3a.** Standardbeispiel ohne Terminierung: VA für gestörte Codefolge

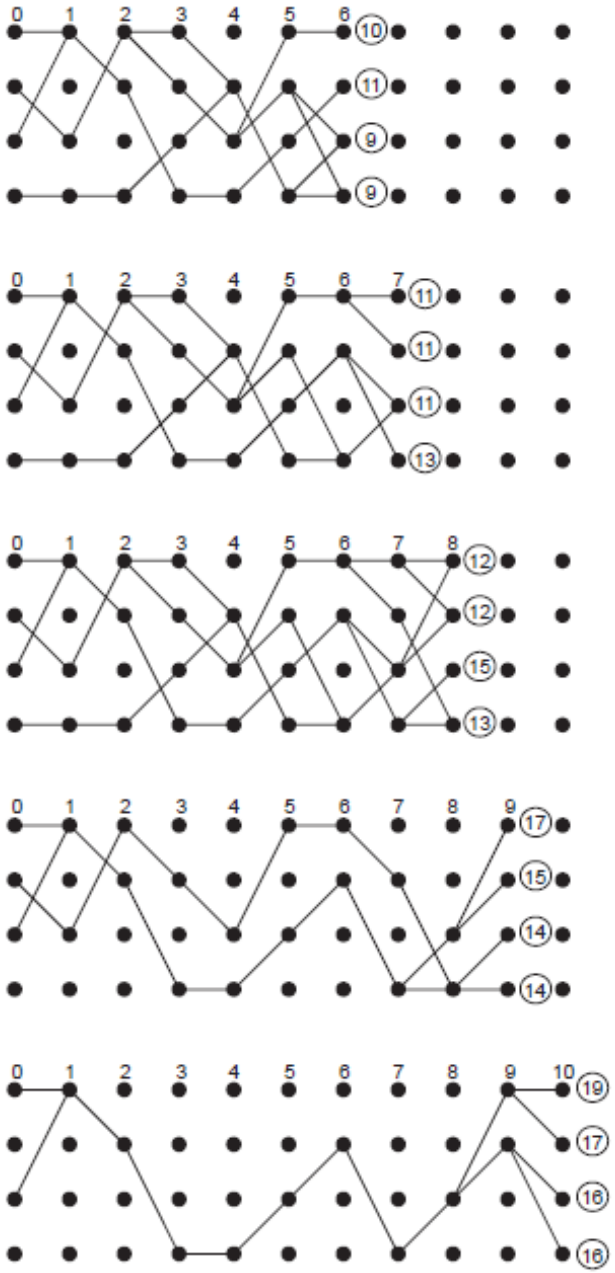


**Bild 9.3b.** Fortsetzung von Bild 9.3a

Für dieses Beispiel:

- Nach dem Segment (Codeblock) Nr.9 kann eine finale Schätzung für Segment Nr.0 fallen, denn die weitere Fortsetzung der Survivor hat keine Rückwirkung auf den Anfangsbereich der Survivor.
- Nebenbei: im Segment Nr.0 gibt es keine Mehrdeutigkeit da beides obere Kanten sind.

Allgemein:

- Zur Zeit r kann eine endültige Schätzung für die Zeit r-v erfolgen.
- v = Rückgrifftiefe
- Faustregel: $v \approx 5 \cdot m$ (Details folgen)

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 Infobit $u_{r-v}$ 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 Codierungsgewinn. 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.

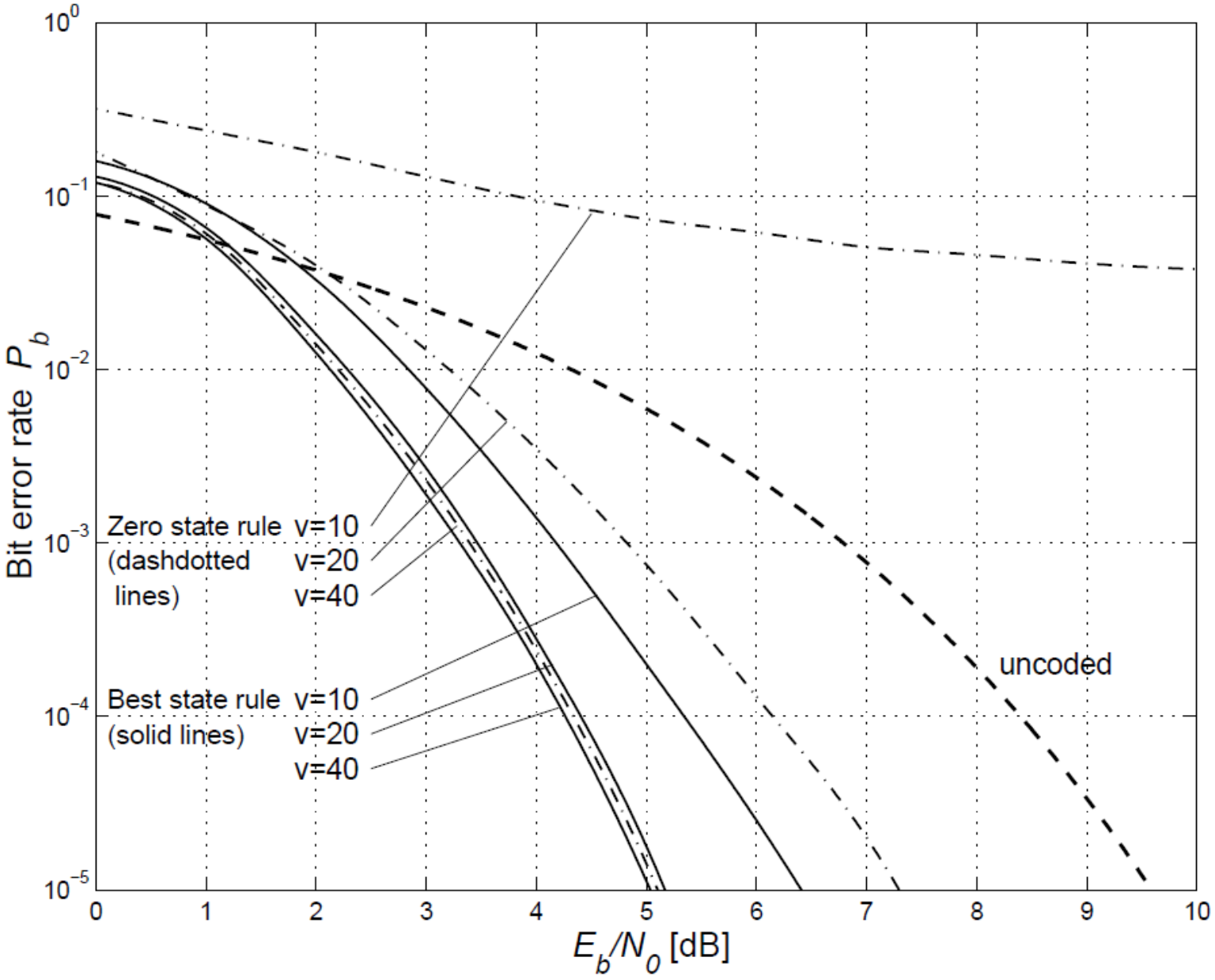



**Bild 9.4.** Vergleich der Entscheidungsstrategien beim VA ($R = 1/2, m = 4$)

Entscheider-Strategien:

- Zero State Rule = Rückverfolgung von einem fest eingestelltem Zustand
- Best State Rule = Rückverfolgung vom Zustand mit maximaler Metrik

Fazit (bei m=4)

- Bei kleinem v (hier v=20) ist Zero State Rule noch deutlich schlechter als Best State Rule.
- Bei großem v (hier v=40) sind beide Strategien ähnlich und können durch weitere Vergrößerung von v nicht weiter verbessert werden.

Anmerkung: Typischerweise werden Aussagen...

- bei Blockcodes mathematisch bewiesen
- bei Faltungscodes heuristisch durch Probieren und Simulation bestätigt

Aufgliederung des Viterbi-Decoders in

- Metrik-Prozessor: unabhängig vom Code, aber abhängig von der Punktierung, maximal $\min\{2^n, 2^m\}$ verschiedene Inkremente
- ACS-Prozessor (Add-Compare-Select): nur hier geht der Code ein (Zuordnung Codeblöcke zu Kanten)

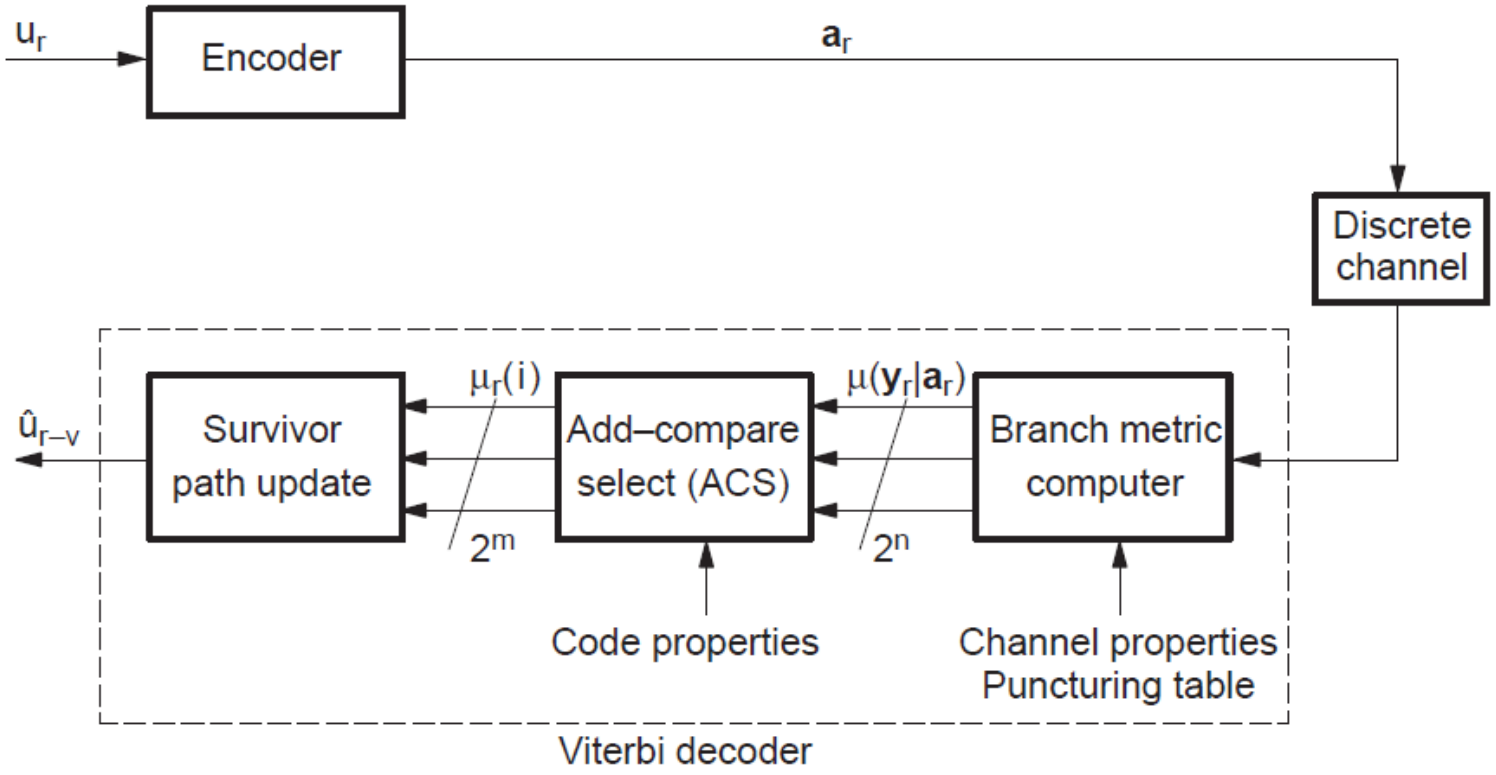



**Bild 9.5.** Gliederung des Viterbi-Decoders

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).

Thema
Quantisierung

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).

Thema
Punktierung

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.

Thema
Speicher vs. Operationen

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 [3, 19, 75].

Thema Zahlbereichs-überschreitung

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:

$$\text{Ersetze} \quad \mu_r(i) \quad \text{durch} \quad \mu_r(i) - C \quad \text{für} \quad i = 1, \ldots, 2^m. \tag{9.4.1}$$

Thema Anfangszustand

*Knotensynchronisation:* Sofern nicht der Empfänger in eine laufende Übertragung eingeschaltet wird, tritt ein Anfangseffekt auf wie bereits mit Bild 8.8 erklärt wurde. Der Start beim Nullzustand kann im ACS-Prozessor automatisch dadurch berücksichtigt werden, daß die Survivor-Metriken zu Beginn mit

$$\mu_0(1) = 0 \quad , \quad \mu_0(2) = \cdots = \mu_0(2^m) = -\infty \tag{9.4.3}$$

(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.

Thema Synchronisation der Codeblöcke

*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 Infobits 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 Faltungscodes wird so entdeckt.

Thema Rahmen-Synchronisation (weitere Details folgen noch bei RS*CC)

*Rahmensynchronisation bei terminierten Codes:* Dies wird bei den meisten Anwendungen durch Maßnahmen außerhalb der Kanalcodierung erreicht, beispielsweise durch uncodierte Synchronisationswörter. 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 Blockcodes.

```
function uhat = fct_cc_viterbidec_bin(z, n, m, L, gpol, rule, vback);

% n = 1/coderate
% m = memory length
% L = number of information bits
% uhat      (1,    L+m)        estimated infobits with termination, range={0,1}
% z         (1,    n(L+m))     received symbols,                    range={-1,+1}+R
% metold    (1,    2^m)        survivor metrics,                    range=R
% metnew    (1,    2^m)        survivor metrics,                    range=R
% metrikall(L+m+1,2^m)        gesamte Metrik (für best state rule)
% surv      (2^m,  L+m+1)      survivor (info bit),                 range={0,1}
% codeblock(n,    2^(m-1))    codeblock at branch from 1 to 2*1-1, range={-1,+1}
% inc       (1,    2^(m-1))    increment at branch from 1 to 2*1-1, range=R
% survtrace(1,    L+m+1)      survivor states at traceback,        range={1...2^m}
% gpol      (n,    1+m)        generator polynomial,                range={0,1}
% vback                       chainback in segments
% rule =0(zero) =1(best)
zm            = 2^m;
zm2           = 2^(m-1);
surv          = zeros(zm,L+m+1);
surv(:,1)     = 999;
metold(1)     = 0;
metold(2:zm)  = -10000;
uhat          = zeros(1,L+m);
survtrace     = zeros(1,L+m+1);
%-------------------------------------------------------------------
% labeling codeblock for branch from ls to 2*ls-1
% codeblock[ls,ln]=0*g_ln[0]+u_r*g_ln[1]+...+u_{r-m}*g_ln[m]
%                 = ustring[m]*gpol[ln,2] + ... + ustring[1]*gpol[ln,m+1]
for ls = 1:zm2
   ustring(1:m) = dec2bin(ls-1, m);     % dual number representation as string
   for lm=1:m
      u(lm) = str2num(ustring(m+1-lm)); % dual number representation as number
   end
   codeblock(1:n,ls) = 1 - 2*mod( gpol(1:n,2:m+1)*transpose(u) , 2);
end
%-------------------------------------------------------------------
for lseg = 1:L+m
   inc = z(n*(lseg-1)+1:n*lseg) * codeblock;
   metnew([1:2:zm-1]) = max(metold(1:zm2)+inc, metold(zm2+1:zm)-inc);
   metnew([2:2:zm  ]) = max(metold(1:zm2)-inc, metold(zm2+1:zm)+inc);

   surv([1:2:zm-1],lseg+1) = transpose([metold(1:zm2)+inc < metold(zm2+1:zm)-inc]);
   surv([2:2:zm  ],lseg+1) = transpose([metold(1:zm2)-inc < metold(zm2+1:zm)+inc]);

   metold = metnew;
   metrikall(lseg+1,:) = metnew;
end
metall(1,:) = zeros(1,zm);
%-------------------------------------------------------------------
for time = 1:L+m
   timestart = min(time+vback, L+m+1);
   if (rule==0)
      survtrace(timestart) = 1;
   else
      [wert,position]      = max(metrikall(timestart,:));
      survtrace(timestart) = position;
   end
   for lseg = timestart-1 : -1 : time
      currentstate    = survtrace(lseg+1);
      survtrace(lseg) = ceil(currentstate/2) + surv(currentstate,lseg+1)*zm2;
   end
   uhat(time) = 1-mod(currentstate,2);
end
```

Komplette Matlab-Funktion zur Realisierung eines Viterbi-Decoders
(optimiert auf geringe Rechenzeit, hoher Speicherbedarf)

Auch wenn es eine fertige Routine im Matlab Communications Paket gibt zeigt dieses Beispiel doch:
Die Implementierung des Viterbi-Algorithmus ist simpel – im krassen Gegensatz zum RS-Decoder!

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

$$\boldsymbol{e}_r = \boldsymbol{a}_r - \hat{\boldsymbol{a}}_r \tag{9.5.1}$$

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.

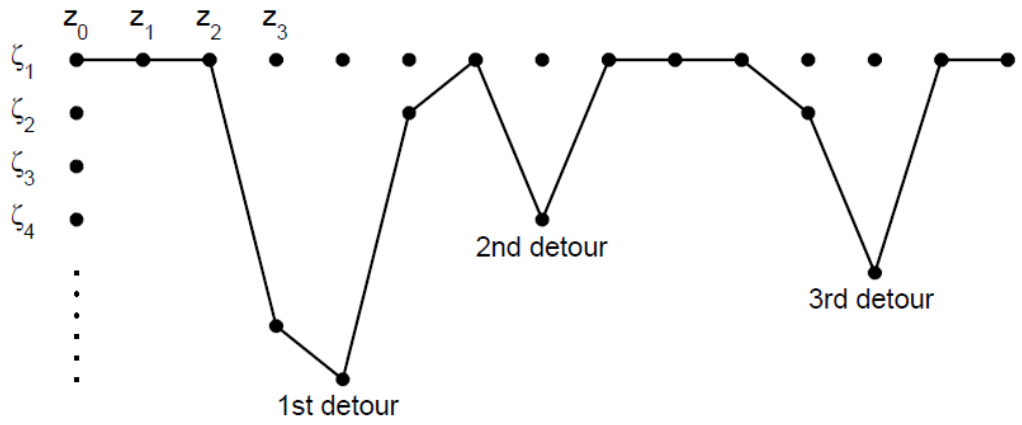



**Bild 9.6.** Fehlerfolge interpretiert als Abfolge von Fundamentalwegen

**Satz 9.1.** *Für die asymptotischen Codierungsgewinne gilt (in dB):*

$$G_{\mathrm{a,hard}} = 10 \cdot \log_{10}\left(\frac{Rd_{\mathrm{f}}}{2}\right) \quad , \quad G_{\mathrm{a,soft}} = 10 \cdot \log_{10}\left(Rd_{\mathrm{f}}\right). \tag{9.5.4}$$

*Soft-Decision bringt gegenüber Hard-Decision einen Gewinn von 3 dB wie bei den Blockcodes.*

Zur Erinnerung:
$G_a$ bei Blockcodes

$$G_{\mathrm{a,hard}} = 10 \cdot \log_{10}(R(t+1)) \quad \mathrm{dB}. \qquad G_{\mathrm{a,soft}} = 10 \cdot \log_{10}(Rd_{\mathrm{min}}) \quad \mathrm{dB}.$$

Erhöhung von m um 1 bringt jeweils etwa 0.4 dB Gewinn,
kostet aber doppelten Aufwand im Viterbi-Decoder

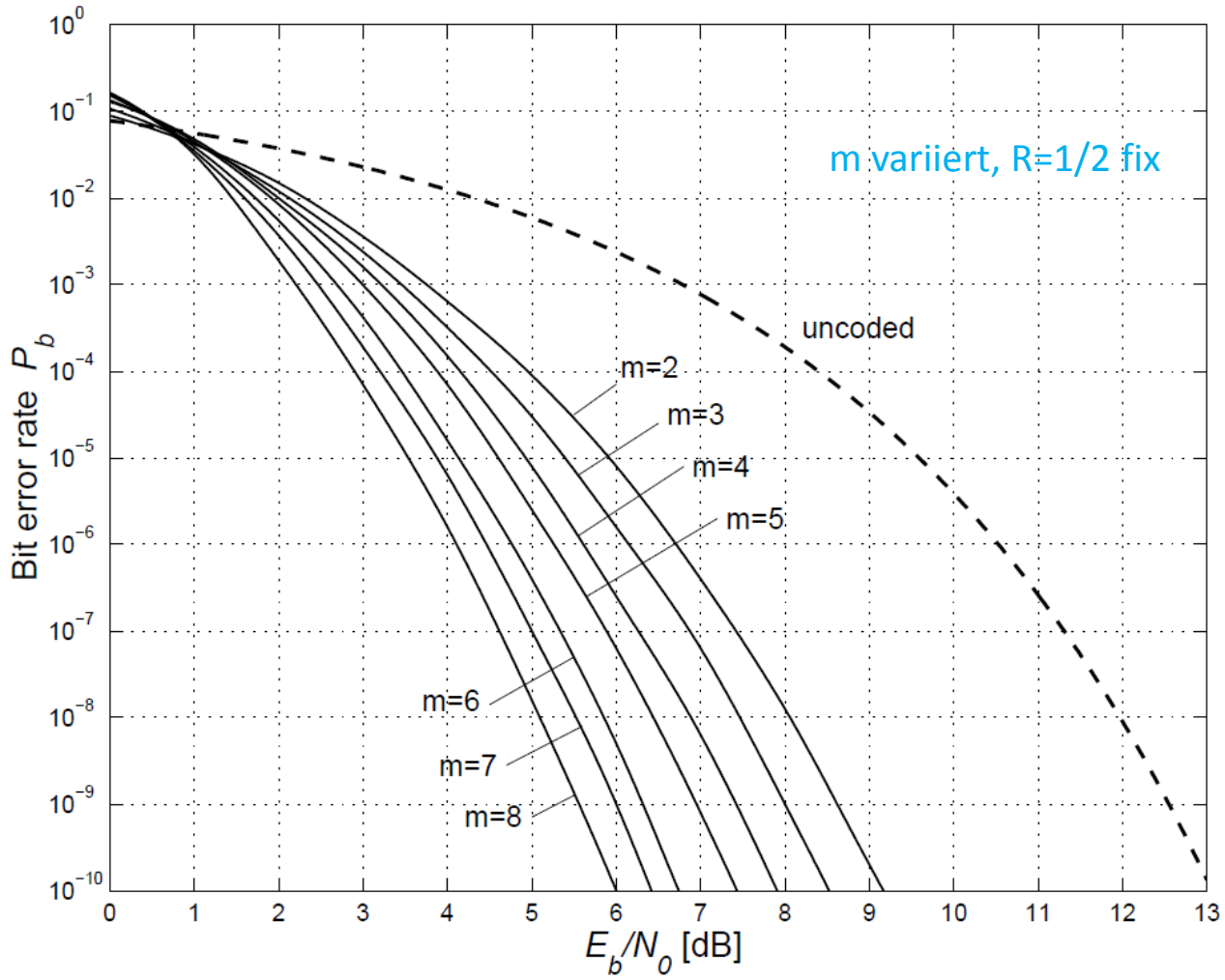



**Bild 9.7.** Fehlerwahrscheinlichkeit der $R=1/2$-Codes bei $m = 2, \ldots, 8$

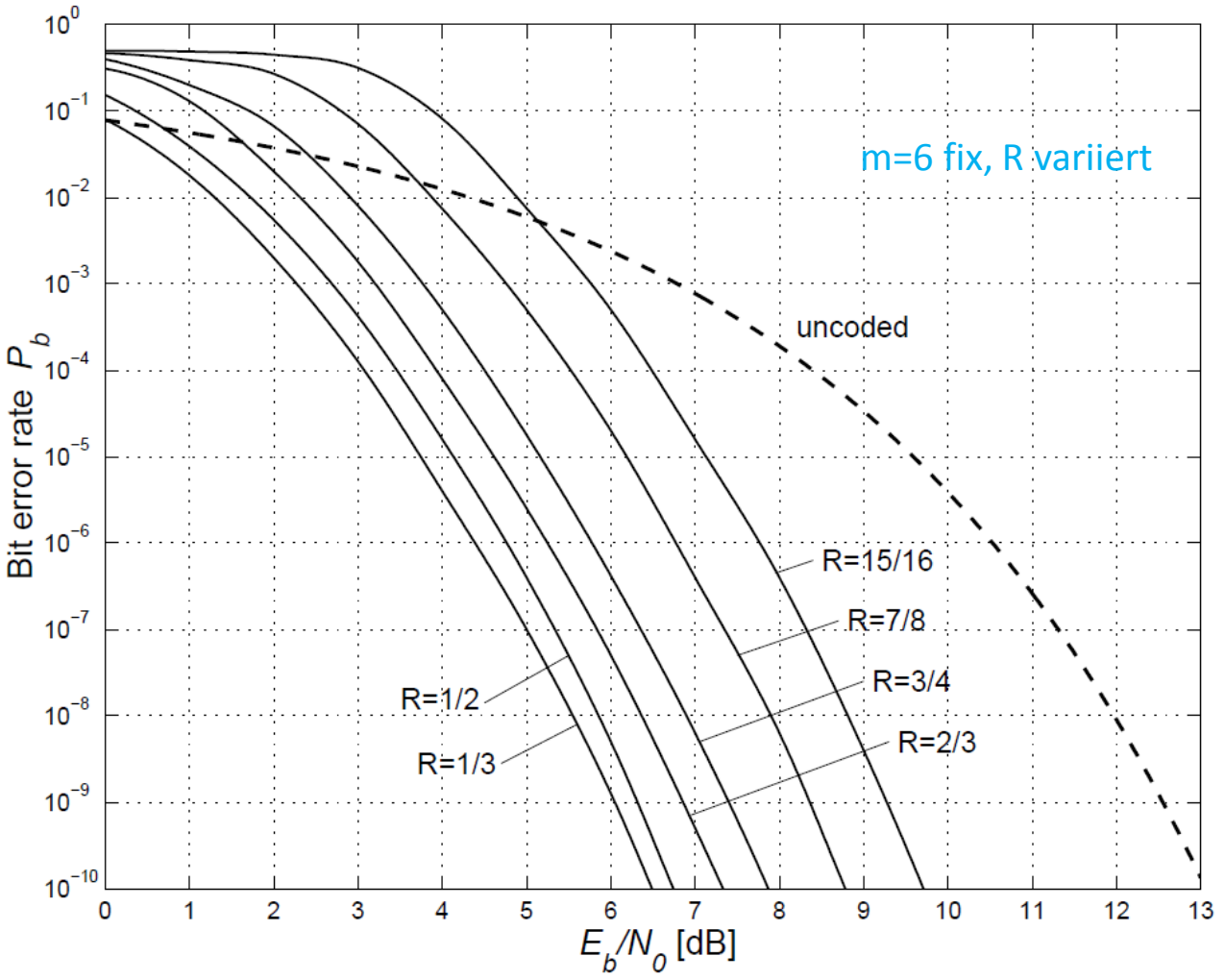



**Bild 9.8.** Fehlerwahrscheinlichkeit der $m=6$-Codes bei verschiedenen Coderaten

Voraussetzungen: AWGN-Kanal, Soft-Decision, ausreichend große Rückgrifftiefe

**Tabelle 9.2.** Asymptotische Codierungsgewinne optimaler Faltungscodes

| $m$ | $R = 1/2$ | | | $R = 1/3$ | | |
|---|---|---|---|---|---|---|
| | $d_f$ | $G_{a,hard}$ [dB] | $G_{a,soft}$ [dB] | $d_f$ | $G_{a,hard}$ [dB] | $G_{a,soft}$ [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 |

**Tabelle 9.3.** Codierungsgewinne bei moderaten Fehlerraten (nach [114])

| $P_b$ | Notwendiges $E_b/N_0$ [dB] für $P_b$ uncodiert | Codierungsgewinn [dB] bei $P_b$ Soft-Decision, $m = 6$ | |
|---|---|---|---|
| | | $R = 1/2$ | $R = 1/3$ |
| $10^{-3}$ | 6,8 | 3,8 | 4,2 |
| $10^{-5}$ | 9,6 | 5,1 | 5,7 |
| $10^{-7}$ | 11,3 | 5,8 | 6,2 |
| asymptotisch | | 7,0 | 7,0 |

Vergleich von R=1/2 und R=1/3 bei m=6:

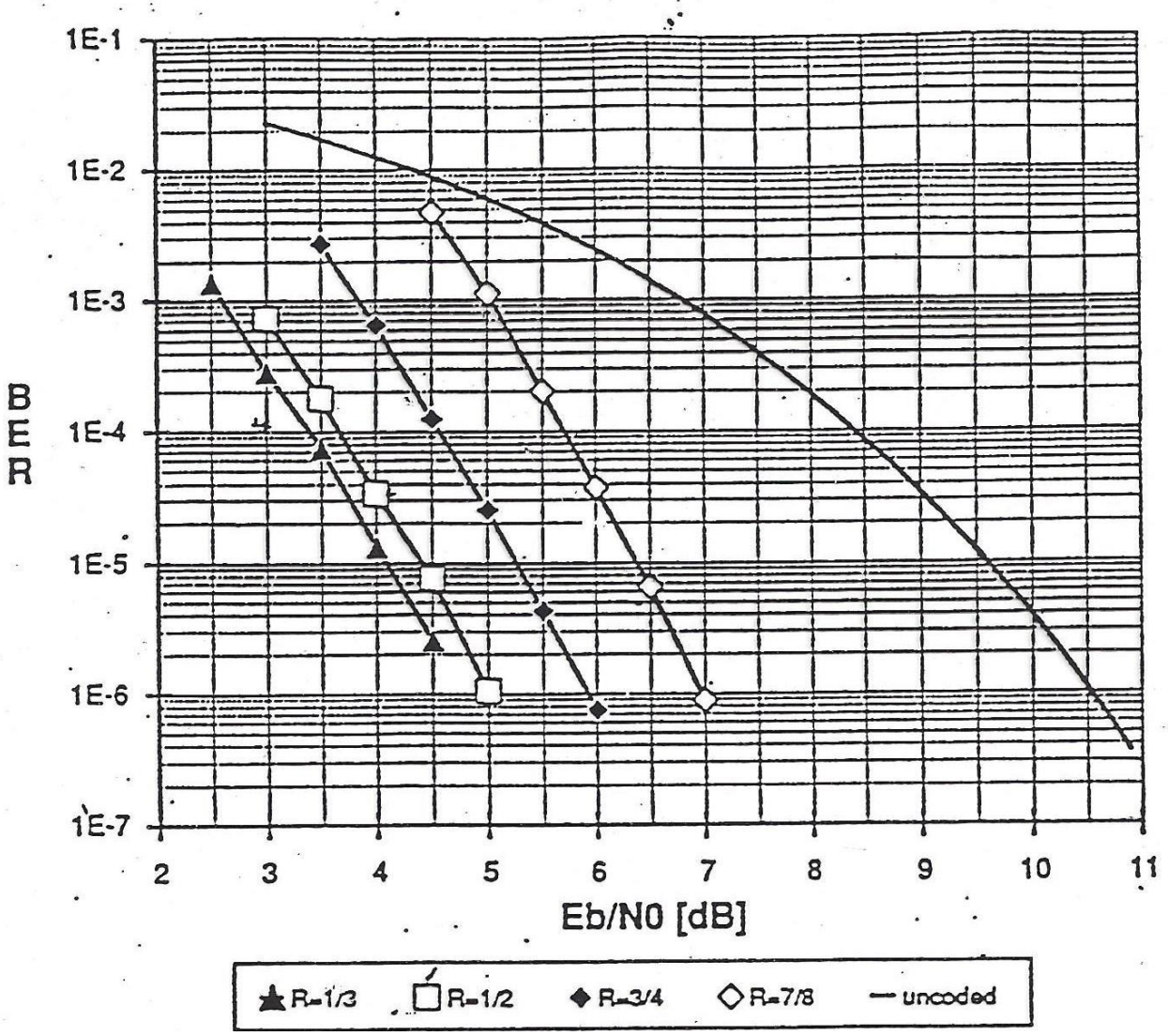
- gleiches $G_a$
- bei $10^{-10}$ noch 0.25 dB Differenz (Bild 9.8)
- bei $10^{-5}$ noch 0.6 dB Differenz (Tabelle 9.3)

Beachte: der Decoder-Aufwand ist ziemlich unabhängig R, aber die kleinere Rate R=1/3 erfordert 50% mehr Bandbreite als R=1/2

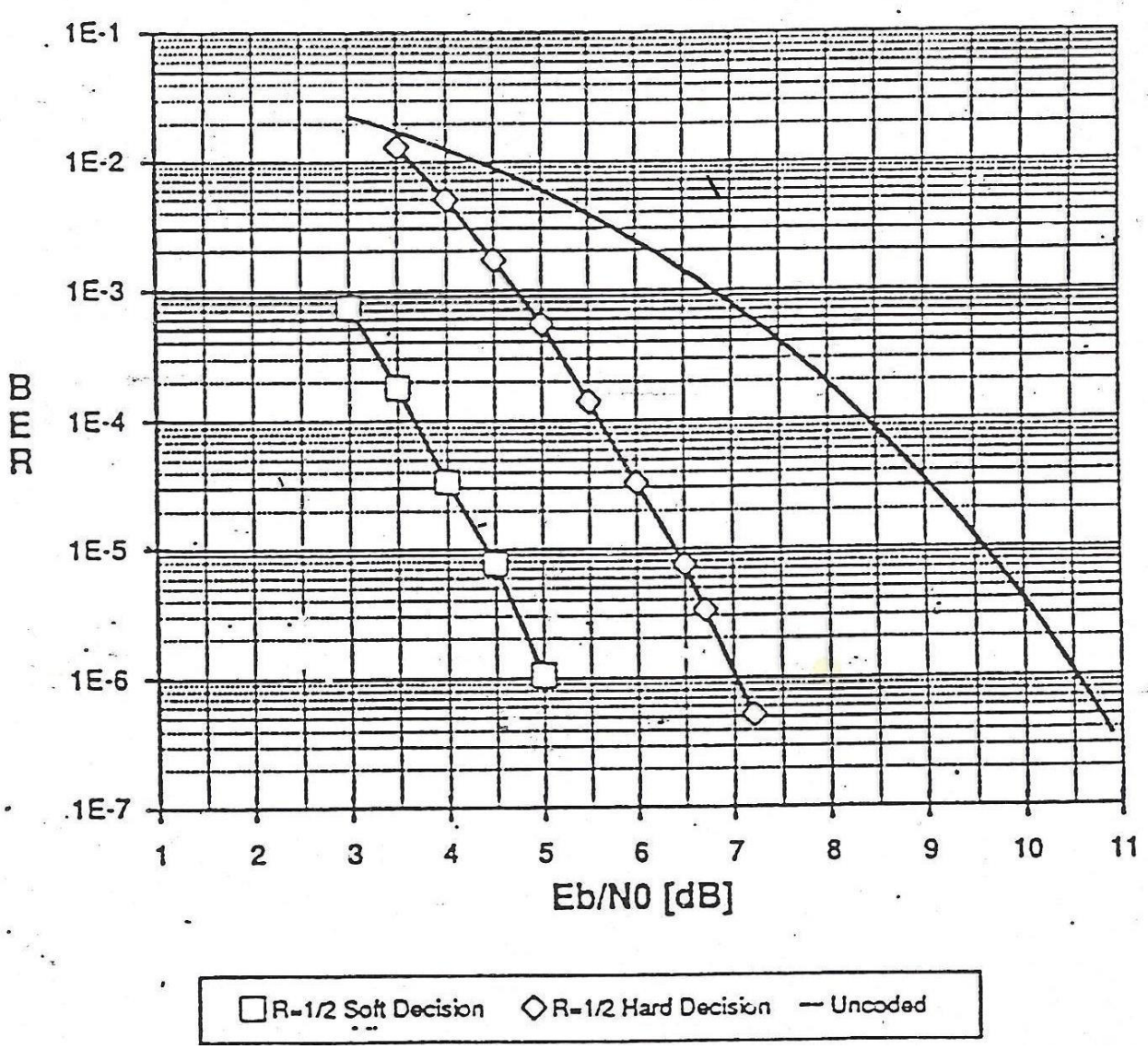
Bei Hard-Decision verschieben sich die Kurven nach rechts, und zwar um ca. 2,2 dB bei moderaten Fehlerraten bzw. um 3 dB im asymptotischen Grenzfall. Bei oktaler Quantisierung mit 3 Bit verschieben sich die Kurven nur um ca. 0,2 dB nach rechts

# Fehlerwahrscheinlichkeit des Industriestandard-Codes [Qualcomm]

## Verschiedene Coderaten




## Soft- versus Hard-Decision



In Bild 9.9 sind die Codierungsgewinne $G_{\text{soft}}$ 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.

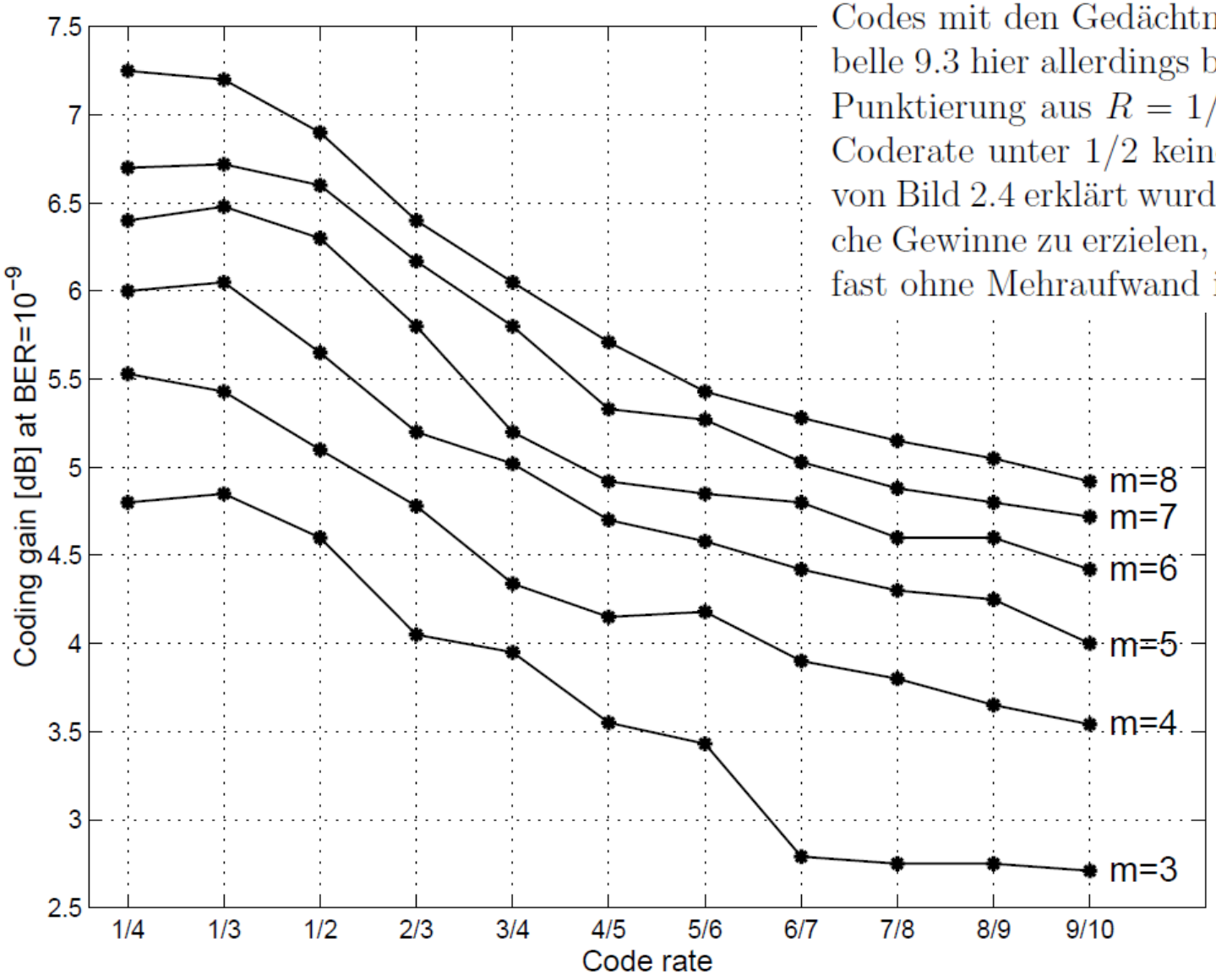



**Bild 9.9.** Codierungsgewinne für $R = 1/4 \ldots 9/10$ und $m = 3 \ldots 8$ (nach [119])

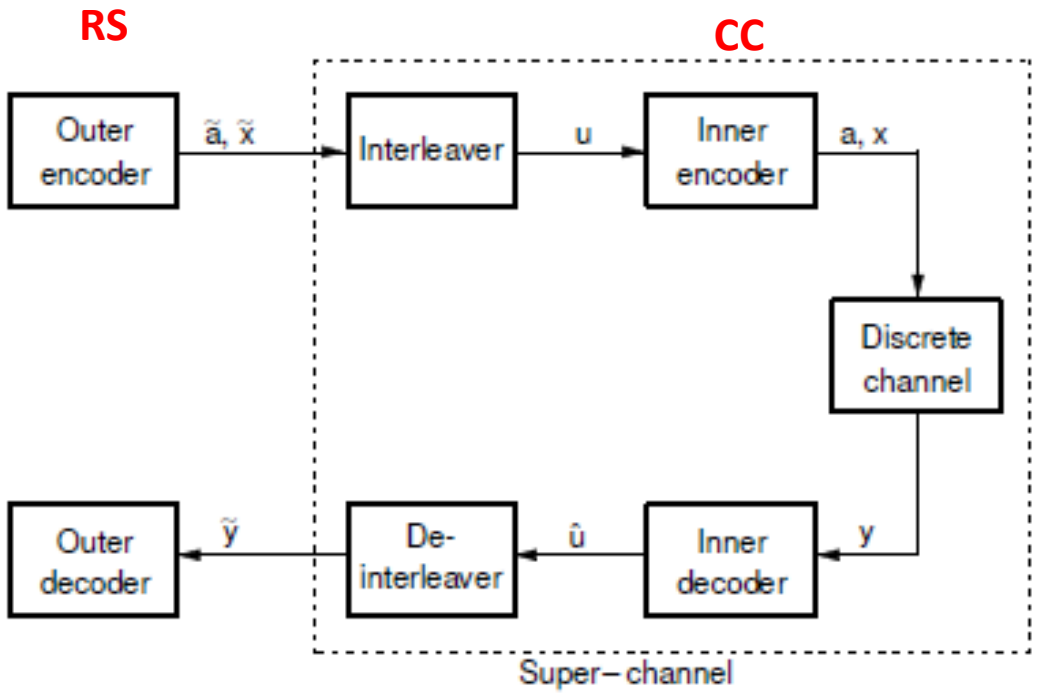



**Figure 10.16.** Concatenated coding

**Table 10.4.** Typical applications of concatenated coding

| Innerer Code | Äußerer Code |
|---|---|
| block code | block code |
| convolutional code | RS-Code |
| MLSE-Entzerrer | convolutional code, TCM |
| convolutional code | Quellencode |
| convolutional code | convolutional code |

Anwendung u.a CD
**Wichtigster Fall RS*CC**

Mobilfunk, aber nicht verwirklicht
Turbo-Codes

Nach der Viterbi-Decodierung von Faltungscodes treten Bündelfehler auf

- die Bündelfehler sind eine Eigenschaft des Codes und nicht des Viterbi-Algorithmus
- der Superkanal mit CC innen ist das wichtigste Beispiel eines Bündelfehler-erzeugenden Kanals
- die Bündelfehler prägen sowohl den Entwurf von Codierungssystemen wie auch die End- Anwendung (SLA = Service Level Agreement) ganz wesentlich

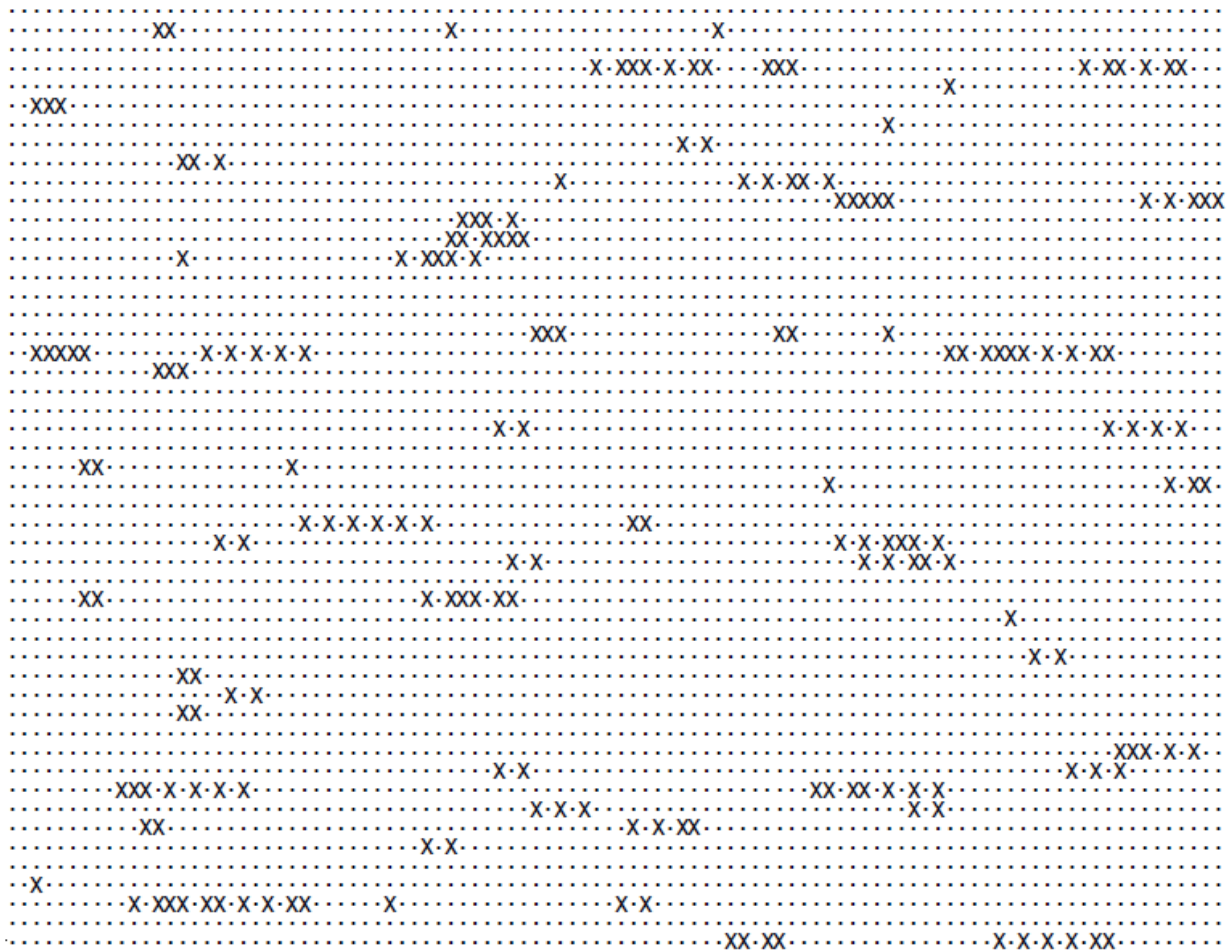
Die nächsten 4 Bilder zeigen Einzelfehler ohne Codierung sowie Bündelfehler am CC-Ausgang bei m=2 und m=6 und bei Punktierung

- Alle Bilder enthalten 50 x 100 = 5000 Bits
- ein Kreuz markiert einen Fehler
- Die Fehlerrate in allen Bildern beträgt etwa 0.04, also enthält jedes Bild etwa 200 Fehler.
- Die hohe Fehlerrate und das gewählte $E_b/N_0$ sind zwar eher unrealistisch, erlauben jedoch einen optisch eindrucksvollen Vergleich.

## Einzelfehler eines BSC

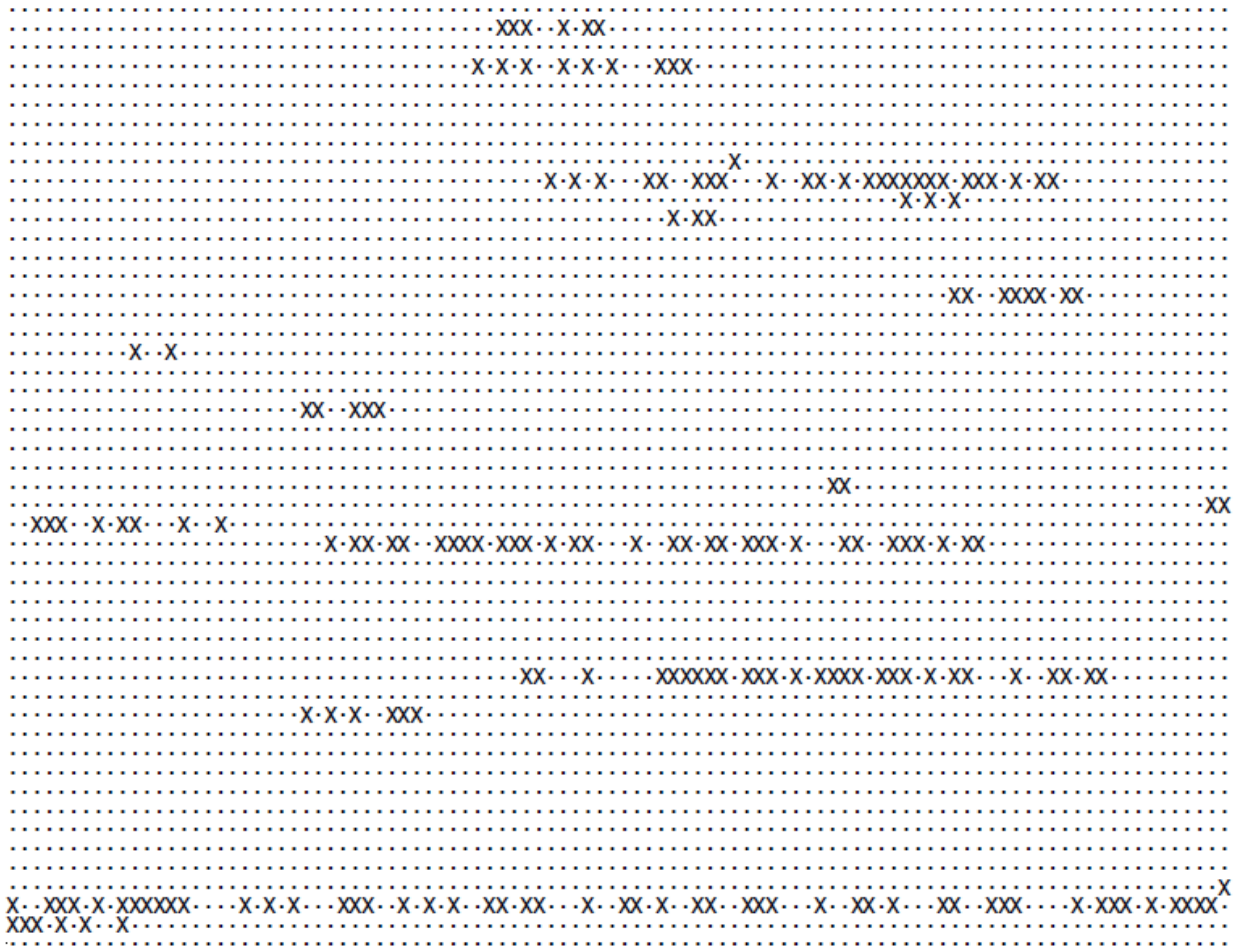
Die statistische Unabhängigkeit der Einzelfehler wird deutlich.

## Bündelfehler nach dem Decoder CC(R=1/2,m=2,soft) @ $E_bN_0$=1.1 dB



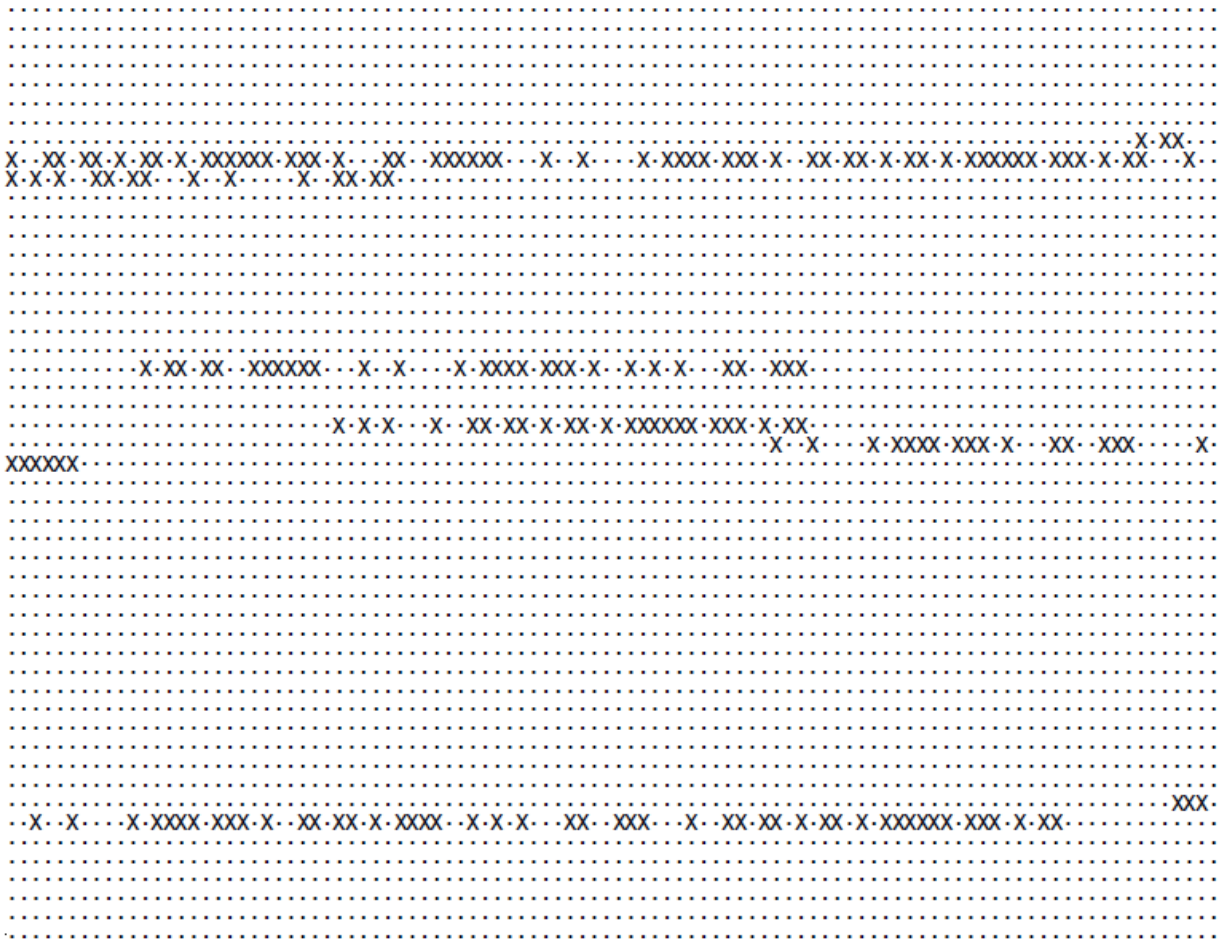
Hier sind lange und kurze Bündelfehler wie auch Einzelfehler zu beobachten.

## Lange Bündelfehler nach dem Decoder
## CC(R=1/2,m=6,soft) @ $E_bN_0$=1.1 dB



Größeres m bedeutet längere Bündelfehler, durch Punktierung werden die Bündelfehler noch länger.

## Sehr lange Bündelfehler nach dem Decoder
## CC(R=7/8,m=6,punctured,soft) @ $E_bN_0$=3.5 dB



Hier gibt es scheinbar überhaupt keine Einzelfehler mehr, dafür aber tritt hier ein Bündelfehler der Länge 150 auf.

Lange Bündelfehler haben eine üble Auswirkung auf die Simulation von Kanalcodierung:

Um eine Bitfehlerrate einigermaßen zuverlässig schätzen zu können, sollten etwa 10 ... 100 Fehler beobachtet werden

Für BER = $p_b = 10^{-5}$ bedeutet das:

- bei Einzelfehlern sind mindestens 10 ... 100 Einzelfehler zu beobachten
  $\Rightarrow$ Simulationsdauer = $10^6$ ... $10^7$ Infobits

- bei Bündelfehlern der Länge 100 sind mindestens 10 ... 100 Bündelfehler zu beobachten
  $\Rightarrow$ Simulationsdauer = $10^8$ ... $10^9$ Infobits

Für BER = $p_b = 10^{-8}$ erhöht sich die Simulationsdauer nochmals um den Faktor 1000.

Da helfen meistens auch keine schnelleren Rechner sondern eher tiefgehendere mathematische Analysen.

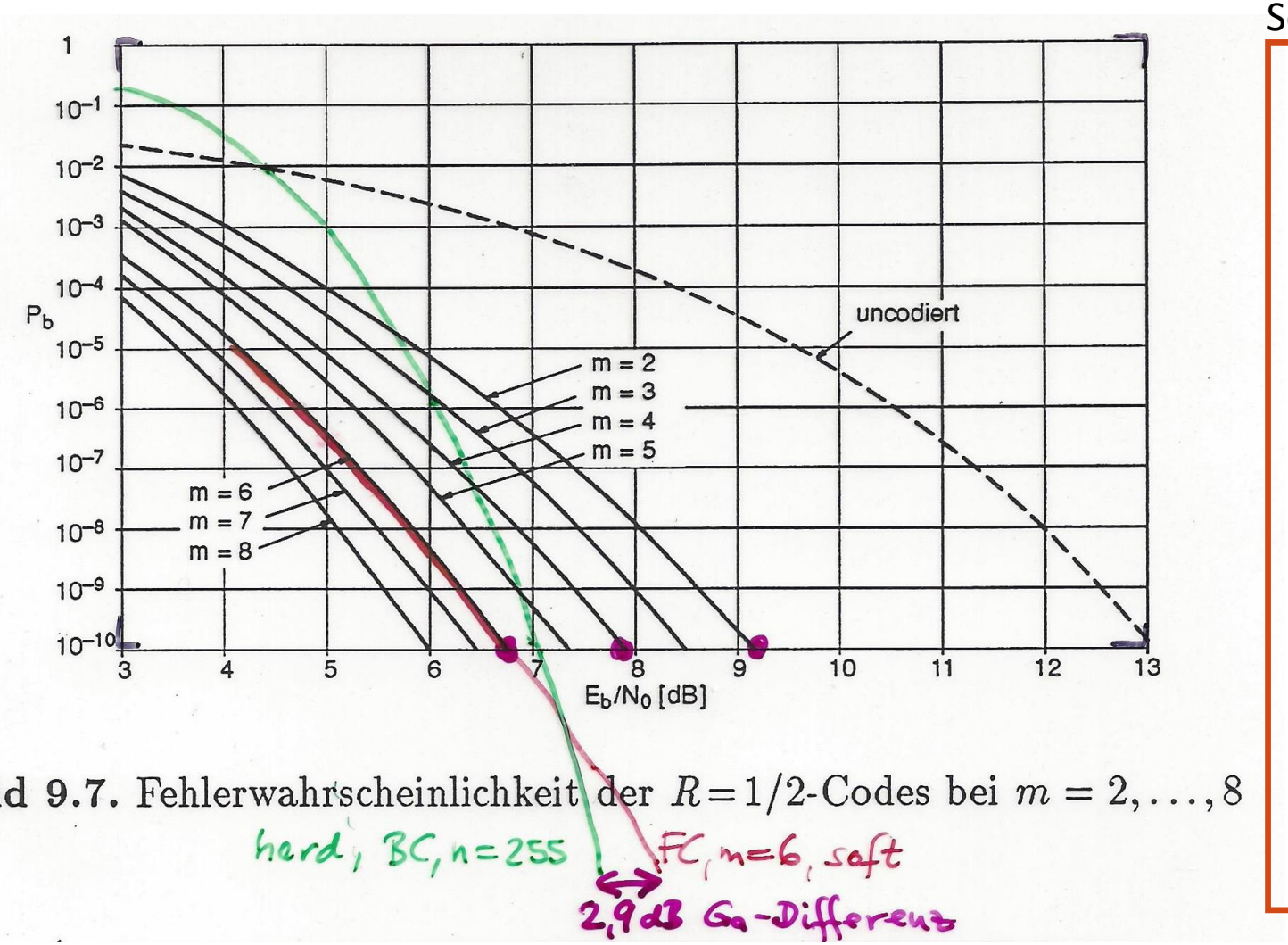



**Bild 9.7.** Fehlerwahrscheinlichkeit der $R = 1/2$-Codes bei $m = 2, \ldots, 8$

Sehr wichtige Themen: Fairness, Steilheit

Vergleich von
- Faltungscode CC(m=6, R=1/2), Soft-Decision
- Blockcode BCH(255,131,t=18)$_2$, Hard-Decision

Der Vergleich ist fair:
- R ähnlich
- Decoder-Aufwand ähnlich
- CC mit Soft / BCH mit Hard

Fazit
- CC besser bei schlechten Kanälen, BCH besser bei sehr guten Kanälen
- CC hat eine flachere BER-Kurve, das ist unter praktischen Gesichtspunkten besser: falls der Kanal degradiert, ist der BER-Anstieg nicht so steil wie bei BCH.

Abkürzungen
FC = CC = Faltungscode = Convolutional Code
BC = Blockcode

**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 Hard-Dec. | $R = 1/2$-convolutional code mit $m = 6$ $d_f = 10$, ML-Decoder 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 |

Diese Tabelle enthält wie auf der vorigen Seite den Vergleich von

- Faltungscode CC(m=6, R=1/2), Soft-Decision
- Blockcode BCH(255,131,t=18)$_2$, Hard-Decision

jetzt aber zusätzlich auch noch die Variante

- Faltungscode CC(m=6, R=1/2), Hard-Decision

Während CC-Soft deutlich besser als BCH-Hard ist (abgesehen von sehr kleinen Fehlerraten),
ist CC-Hard gegenüber BCH-Hard nicht attraktiv, das folgt sowohl aus obiger Tabelle sowie aus nachfolgender Grafik.

Die Variante BCH-Soft ist nicht realisierbar und taucht deshalb nicht auf.

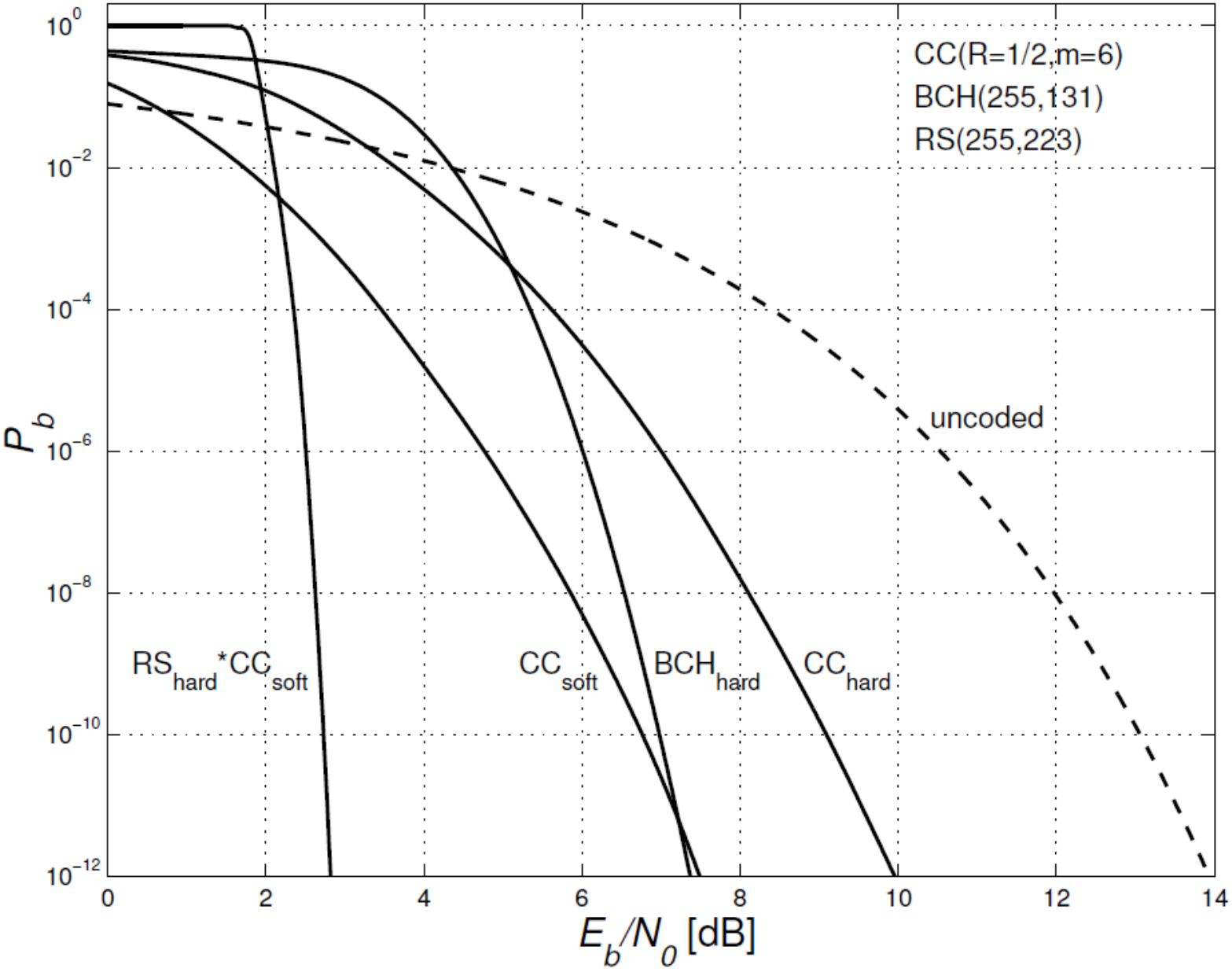



**Figure 10.17.** Performance comparison of block and convolutional codes

Nochmals der (faire) Vergleich von

- Faltungscode CC(m=6, R=1/2), Soft
- Faltungscode CC(m=6, R=1/2), Hard
- Blockcode BCH(255,131,t=18)$_2$, Hard
- Verkettung $RS_{hard}$*$CC_{soft}$

Die Codeverkettung RS*CC wird noch intensiv behandelt (zur Erinnerung: die RS*RS Verkettung wurde bei der Compact Disc Digital Audio verwendet). Hier nur der Hinweis auf die extreme Steilheit der Kurve, d.h. nach der Decodierung ist entweder alles fehlerfrei oder total falsch, der Zwischenbereich ist extrem schmal.

| Kriterium | Blockcodes | Faltungscodes |
|---|---|---|
| Historie | zu Beginn der Informationstheorie überwiegend verwendet, Beweis der Codierungstheoreme | werden inzwischen gleichoft verwendet wie Blockcodes |
| Mathematik | weit entwickelte und geschlossene Theorie, teilweise sehr anspruchsvoll | Theorie mit der gleichen Aussagekraft wie bei Blockcodes ist noch schwieriger, wird aber selten benötigt |
| $\mathbb{F}_{p^m}$ | $p = 2$ (in Sonderfällen $p > 2$)<br>$m \leq 10$ bei RS-Codes<br>$m \leq 10 \ldots 14$ bei BCH-Codes | $p = 2$<br>$m = 1$ |
| Grundstruktur | Vektorräume (Matrizen über Körpern):<br>Intensive Kenntnisse der Galoisfeld-Arithmetik erforderlich bei Entwicklung und Implementierung hochentwickelter Verfahren | Module (Matrizen über Polynomringen):<br>Intensive Kenntnisse der Modul-Theorie erforderlich bei Umformung von $(n, k)$-Encodern, aber praktisch selten notwendig |
| Beziehung zueinander (formal) | Blockcodes sind spezielle Faltungscodes ohne Gedächtnis | Faltungscodes sind verallgemeinerte Blockcodes und terminierte Faltungscodes sind spezielle Blockcodes |

| Kriterium | Blockcodes | Faltungscodes |
|---|---|---|
| Coderate | $R = k/n$ nahezu beliebig zwischen 0 und 1 | $R = 1/n$,<br>$R \to 1$ mit Punktierung |
| Gute Codes durch | große Blocklänge $n$ | große Gedächtnislänge $m$ |
| Leistungsfähige Codes | können als parametrische Klasse analytisch geschlossen angegeben werden (RS, BCH) | werden durch Rechnersuche gewonnen, sind nicht theoretisch ableitbar |
| Gütekriterium | Mimimaldistanz, Gewichtsverteilung | freie Distanz, Gewichtsverteilung |
| Polynombeschreibung | 1 Generatorpolynom vom Grad $n - k$ (bei zyklischen Codes) | $k \cdot n$ Generatorpolynome vom Grad $\leq m$ (typisch $k = 1$) |
| weitere Charakterisierungen | spektral (DFT) | Trellis-, Zustands-, Baumdiagramm |
| Systematische Codes | sind immer möglich und werden immer verwendet | haben generelle Nachteile und werden nur in Sonderfällen verwendet (Ausnahme: rückgekoppelte Encoder bei TCM) |

| Kriterium | Blockcodes | Faltungscodes |
|---|---|---|
| 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 |

| Kriterium | Blockcodes | Faltungscodes |
|---|---|---|
| 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<br>Fadingkanäle (Rayleigh, Rice)<br>bandbegrenzte Kanäle<br>Kanäle mit Verzerrungen<br>UEP-Anwendungen | |