Vorlesungsskript Kanalcodierung II

von
DR.-ING. VOLKER KÜHN

aktualisiert von
DR.-ING. DIRK WÜBBEN

Fachbereich Physik/Elektrotechnik (FB 1)
Arbeitsbereich Nachrichtentechnik
Postfach 33 04 40
D-28334 Bremen

Version 2.4

(04.04.2011)

Kapitel 1

Verkettete Codes

1.1 Einführung

Motivation
Ziel der Codierungstheorie ist es, die Kanalkapazität von Shannon zu erreichen. Praktische Codes wie die im letzten Semester behandelten Faltungs- und Blockcodes sind weit von dieser theoretischen Grenze entfernt. Ein entscheidender Nachteil von ihnen besteht darin, dass mit zunehmender Leistungsfähigkeit auch der Decodieraufwand ansteigt, meistens wächst er exponentiell an. Daher setzt die praktische Realisierbarkeit der Leistungsfähigkeit dieser Codes schnell Grenzen. Zwar ist es oft nur eine Frage der Zeit, bis die Technologie einen höheren Aufwand bei der Decodierung erlaubt. Trotzdem stellt sich die Frage, ob nur durch Vergrößerung der Einflusslänge bei Faltungscodes bzw. der Blocklänge bei Blockcodes die Kapazität nach Shannon prinzipiell erreicht werden kann.

Einen anderen Weg zur Konstruktion leistungsfähiger FEC-Codes zeigte Forney bereits im Jahr 1966, als er erstmals die Verkettung einfacher Teilcodes vorstellte [For66]. Seit dieser Zeit sind gerade in den vergangenen Jahren enorme Fortschritte zu verzeichnen gewesen. Hervorzuheben ist hier das Jahr 1993, als zum ersten Mal die sogenannten *Turbo-Codes* präsentiert wurden, einer geschickten parallelen Anordnung zweier Faltungscodes, mit denen die Kapazitätsgrenze von Shannon so dicht wie noch nie erreicht wurde (Abstand nur noch 0,5 dB bei einer Fehlerrate von $P_b = 10^{-5}$).

Idee
Die grundlegende Idee besteht also darin, einfache Codes geschickt miteinander zu verknüpfen, so dass ein Gesamtcode entsteht, der leistungsfähiger als die einzelnen Komponentencodes ist. Gleichzeitig muss er aber auch einfacher zu decodieren sein. Dies soll an einem Beispiel näher erläutert werden. Die heute zur Verfügung stehende Technologie erlaubt beispielsweise im Consumerbereich die Decodierung eines Faltungscodes mit der Einflusslänge $L_c = 9$, d.h. es ist ein Trellisdiagramm bestehend aus $2^8 = 256$ Zuständen abzuarbeiten. Würde man nun 2 Faltungscodes mit $L_c = 3$ verknüpfen, ergäbe sich ein Aufwand, der einem Trellisdiagramm von $2 \cdot 2^2 = 8$ Zuständen, also nur der 64-ste Teil des Aufwandes. Berücksichtigt man nun eine wiederholte Decodierung des verketteten Codes (Turbo-Decodierung, z.B. 6 Iterationen, mehr dazu später), ergibt sich ein Aufwand von $6 \cdot 8 = 48$ Zuständen, also immer noch weniger als ein Fünftel des ursprünglichen Aufwandes. Ob der resultierende Gesamtcode genauso gut oder gar besser ist, wird im Folgenden ausführlich diskutiert.

Prinzipiell unterscheiden wir zwei Arten der Codeverkettung.

Serielle Verkettung

- Codes sind seriell hintereinander angeordnet
- Jeder nachfolgende Codierer codiert den gesamten Datenstrom inklusive der bereits erzeugten Redun-

danzbit

- Bei zwei Teilcodes spricht man auch von einem inneren Code (C_2) und einem äußeren Code (C_1)

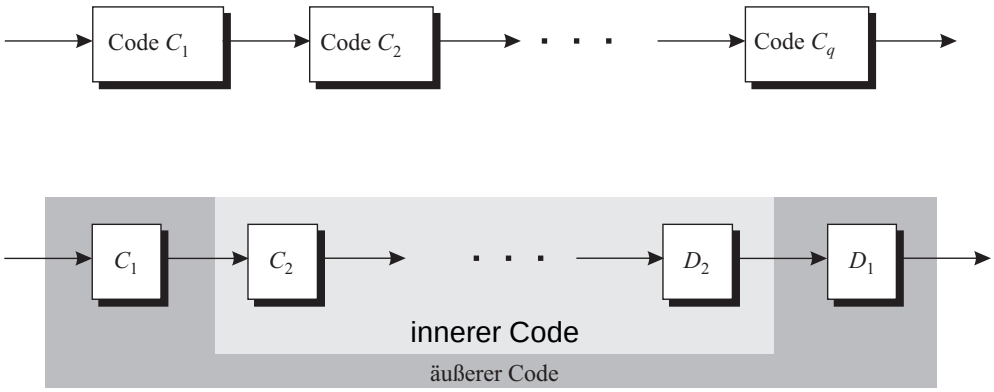

Bild 1.1: Serielle Codeverkettung

Parallele Verkettung

- Teilcodes sind nun parallel angeordnet
- Jeder Teilcodierer erhält nur die Informationsbit, nicht die Redundanzbit der übrigen Codierer
- Die Ausgangssignale der einzelnen Teilcodierer sind durch parallel-seriell-Umsetzung zu einem Datenstrom zusammenzufügen.

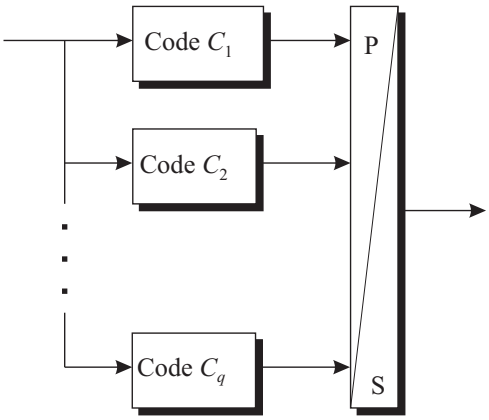

Bild 1.2: Parallele Codeverkettung

In vielen praktischen Systemen sind ebenfalls verkettete Codiersysteme zu finden. So kommen z.B. im GSM-Netz seriell verknüpfte Faltungscodes und CRC-Codes zum Einsatz. Während erstere zur Fehlerkorrektur dienen, bewerkstelligen letztere die Fehlererkennung. Sehr wichtige Kontrollkanäle verwenden anstelle der CRC-Codes auch Fire-Codes zur Fehlerkorrektur.

Bei Weltraummissionen wie beispielsweise der Galileo-Mission findet die Übertragung bei sehr niedrigen Signal-Rausch-Abständen statt, da die Sendeleistung streng begrenzt ist und sehr große Entfernungen zu überbrücken sind. Aus diesem Grund sind hier Codes zu bevorzugen, die unter diesen extremen Randbedingungen gute Ergebnisse erzielen. Das System der NASA verwendet als inneren Code einen Faltungscode, für den mit dem Viterbi-Algorithmus ein effizientes Verfahren zur Soft-Decision-Decodierung zur Verfügung steht. Als äußerer Code wird ein Reed-Solomon-Code eingesetzt, der sehr gut zur Korrektur von Bündelfehlern geeignet ist und somit die Fehlerbündel am Ausgang des Viterbi-Algorithmus korrigieren kann.

Im Rahmen dieser Vorlesung sollen nicht alle Einzelheiten, die bei der Verkettung von Codes eine Rolle spielen behandelt werden. Vielmehr ist es die Aufgabe, ein grundlegendes Verständnis für die grobe Funktionsweise verschiedenen Prinzipien zu entwickeln. Insbesondere die iterative Decodierung in Kapitel 1.7 mit ihren speziellen Decodieralgorithmen ist ein sehr komplexes Feld, das nicht in seiner Vollständigkeit abgehandelt werden kann und noch Gegenstand der aktuellen Forschung ist.

Abschnitt 1.3.2 enthält einen einfachen Einstieg in das Gebiet der verketteten Codes (*concatenated codes*) und erläutert am Beispiel einfacher Produktcodes deren Aufbau und Struktur sowie den Unterschied zwischen serieller und paralleler Verkettung. Danach werden in aller Kürze die erwähnten Turbo-Codes vorgestellt und im Anschluß ein leistungsfähiges Verfahren zur Decodierung verketteter Codes erklärt. Bevor jedoch auf eine geschickte Verkettung einfacher Teilcodes eingegangen wird, erfolgt in nächsten Abschnitt zunächst ein kurzer Einschub zur Erklärung des *Interleaving*.

1.2 *Interleaving* (Codespreizung)

Diese auch als **Codespreizung** oder **Verschachtelung** bekannte Technik beeinflußt in hohem Maße die Leistungsfähigkeit verketteter Codes und stellt somit eine wichtige zu optimierende Komponente dar. Interleaver werden nicht nur in Zusammenhang mit verketteten Codes verwendet, sondern kommen auch bei der Spreizung von Bündelfehlern, die z.B. durch Fading entstanden sind, zum Einsatz.

1.2.1 Blockinterleaver

Der Begriff *Interleaving* beschreibt die Permutation einer Symbolfolge $\mathbf{x}$, d.h. die Veränderung der Reihenfolge der in $\mathbf{x}$ enthaltenen Symbole. Der einfachste Fall ist der sogenannte **Blockinterleaver**, welcher in Bild 1.3 abgebildet ist. In diesem Beispiel besteht er aus 3 Zeilen und 5 Spalten. Er wird Spalte für Spalte mit dem Eingangsvektor

$$\mathbf{x} = (x_0\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_8\ x_9\ x_{10}\ x_{11}\ x_{12}\ x_{13}\ x_{14})$$

beschrieben, allerdings zeilenweise ausgelesen. Somit erhalten wir

$$\tilde{\mathbf{x}} = (x_0\ x_3\ x_6\ x_9\ x_{12}\ x_1\ x_4\ x_7\ x_{10}\ x_{13}\ x_2\ x_5\ x_8\ x_{11}\ x_{14})$$

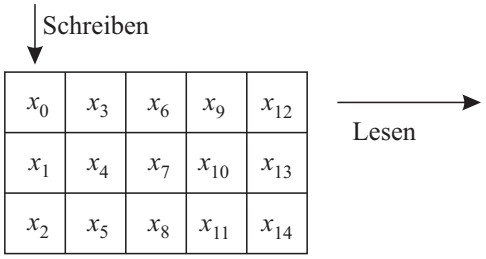

Bild 1.3: Allgemeiner Aufbau eines Interleavers

Es ist zu erkennen, dass zwischen ursprünglich benachbarten Symbolen nun ein Abstand von $L_I = 5$ existiert. Dieser Abstand wird als **Interleavingtiefe** bezeichnet. Die optimale Dimensionierung des Interleavers hängt von mehreren Faktoren ab und wird auch durch den jeweiligen Verwendungszweck beeinflußt.

Anzahl der Spalten
Die Anzahl der Spalten bestimmt direkt die Interleavingtiefe L_I. Sollen beispielsweise Bündelfehler bis zu einer Länge b am Eingang eines Viterbi-Decodierers aufgebrochen werden, muss $L_I \geq b$ gelten, damit sichergestellt ist, dass sich der Bündelfehler in L_I Einzelfehler aufteilt. Dabei ist leicht einzusehen, dass die Interleavingtiefe vergrößert werden kann, wenn die Anzahl der Spalten erhöht wird.

Anzahl der Zeilen
Die zweite Dimension, also die Anzahl der Zeilen, hängt von folgender Betrachtung ab. Bei einem Faltungscode der Einflusslänge $L_c = 5$ sind beispielsweise aufgrund des Gedächtnisses 5 aufeinander folgende Codeworte eng miteinander korreliert. Bei einer Coderate von $R_c = 1/2$ heißt das, dass 10 aufeinander folgende Bit korreliert sind. Sollen diese durch den Interleaver möglichst weit auseinander gespreizt werden (nämlich mit L_I, um sie gut vor Bündelfehlern zu schützen), so muss die Zeilenzahl $L_c/R_c = 10$ betragen. Dann ist sichergestellt, dass jedes dieser 10 Bit zu seinem Nachbarn einen Abstand von L_I besitzt.

Verzögerungszeiten
Dabei gilt zu beachten, dass der Inhalt in der Regel erst ausgelesen werden kann, wenn der Speicherbereich komplett beschrieben worden ist. Hierdurch entsteht eine systembedingte Verzögerungszeit

$$\Delta t = \text{Zeilen} \cdot \text{Spalten} \cdot T_b \tag{1.1}$$

(Regelungstechniker würden von Totzeit sprechen), die kritische Werte nicht überschreiten darf. So gibt es für die Duplex-Sprachübertragung zwar keinen festen Grenzwert einer maximalen Verzögerungszeit, mehr als einige 10 ms gelten aber als nicht tolerierbar. Hieraus folgt, dass z.B. bei einer Datenrate von 9,6 kbit/s eine Interleavergröße von nur etwa 400 bit schon an der oberen Grenze liegt, denn es gilt für Interleaving (Sender) und De-Interleaving (Empfänger)

$$2 \cdot \Delta t = 2 \cdot \frac{400}{9600/s} = 83,3 \text{ ms}.$$

Im GSM-Netz werden für den vollratigen Sprachkanal etwa 22,8 kbit/s übertragen. Für diese Bruttodatenrate ergibt sich bei 400 Bit Interleavergröße eine Verzögerungszeit von

$$2 \cdot \Delta t = 2 \cdot \frac{400}{22.800/s} = 35 \text{ ms}.$$

In der Literatur findet man auch Angaben über maximal zulässige Verzögerungszeiten die wesentlich restriktiver sind (z.B. 20 ms).

1.2.2 Faltungsinterleaving

Auf das sogenannte Faltungsinterleaving soll an dieser Stelle nicht weiter eingegangen werden. Es sei nur soviel gesagt, dass die Funktionsweise der des Blockinterleavings stark ähnelt. Allerdings erreicht man die gleiche Interleavingtiefe schon für geringere Verzögerungszeiten. Für weiterführende Informationen wird auf die Literatur [Fri96] verwiesen.

1.2.3 Zufallsinterleaving

Insbesondere das Blockinterleaving sorgt mit seiner sehr regelmäßigen Struktur dafür, dass Symbole mit einem gewissen Abstand auch nach der Verschachtelung noch den gleichen Abstand zueinander oder aber ein ganzzahliges Vielfaches davon besitzen. Dies führt insbesondere bei den später noch zu behandelnden Turbo-Codes dazu, dass sich schlechte Distanzeigenschaften des Codes ergeben. Daher ist es in diesen Fällen von Vorteil, die Codespreizung quasi-zufällig zu gestalten, d.h. die Art der Permutation ist dem Empfänger natürlich bekannt, sie sollte aber keine systematische Struktur aufweisen. Derartige Interleaver können nicht systematisch konstruiert werden, sondern müssen per Rechnersuche iterativ optimiert werden.

1.3 Serielle Codeverkettung

1.3.1 Vorbetrachtungen

Beispiel 1: Serielle Verkettung von (3,2,2)-SPC-Code und (4,3,2)-SPC-Code
Wir wollen uns anhand eines sehr einfachen Beispiels der geschickten Verkettung zweier Codes nähern. Dazu betrachten wir zunächst einen einfachen (3,2,2)-SPC-Code (*Single-Parity-Check*), der seriell mit einem (4,3,2)-SPC-Code verknüpft werden soll. Die Coderate des verketteten Codes beträgt $R_c = 2/4 = 1/2$, über die Mindestdistanz muss im Folgenden diskutiert werden.

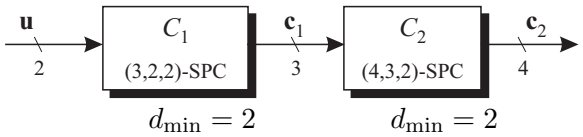

Bild 1.4: Serielle Codeverkettung zweier SPC-Codes

SPC-Codes hängen dem Informationswort $\mathbf{u}$ ein einfaches Paritätsbit an, sie besitzen also die Minimaldistanz $d_{\min} = 2$. Es stellt sich nun die Frage, welche Mindestdistanz die Verkettung hat. Rein anschaulich würde man eine Distanz größer als 2 vermuten. Für dieses einfache Beispiel lässt sich das Distanzspektrum noch anhand einer Tabelle erläutern.

$\mathbf{u}$	$\mathbf{c}_1$	$\mathbf{c}_2$	$w_H(\mathbf{c}_2)$
0 0	0 0 0	0 0 0 0	0
0 1	0 1 1	0 1 1 0	2
1 0	1 0 1	1 0 1 0	2
1 1	1 1 0	1 1 0 0	2

Tabelle 1.1: Codeworte der Verkettung von äußerem (3,2,2)-SPC und innerem (4,3,2)-SPC

Da es sich um einen linearen Code handelt, reicht es aus, die Gewichte der Codewörter zu betrachten und nicht die Distanzen untereinander. Es ist sofort ersichtlich, dass die Mindestdistanz nach wie vor $d_{\min} = 2$ beträgt. Offensichtlich führt eine Verkettung von Codes nicht automatisch zu einer Verbesserung der Distanzeigenschaften. Das folgende Beispiel veranschaulicht, dass bei ungeschickter Kombination zweier Codes der Gesamtcode die freie Distanz des inneren Codes übernimmt.

Beispiel 2: Serielle Verkettung von (4,3,2)-SPC (C_1) und (7,4,3)-Hamming-Code (C_2)
Die Coderate dieser Verkettung lautet $R_c = 3/7$.

$\mathbf{u}$	$\mathbf{c}_1$	$\mathbf{c}_2$	$w_H(\mathbf{c}_2)$	$\tilde{\mathbf{c}}_2$	$w_H(\tilde{\mathbf{c}}_2)$
0 0 0	0 0 0 0	0 0 0 0 0 0 0	0	0 0 0 0 0 0 0	0
0 0 1	0 0 1 1	0 0 1 1 0 0 1	3	0 0 0 1 1 1 1	4
0 1 0	0 1 0 1	0 1 0 1 0 1 0	3	0 1 1 0 0 1 1	4
0 1 1	0 1 1 0	0 1 1 0 0 1 1	4	0 1 1 1 1 0 0	4
1 0 0	1 0 0 1	1 0 0 1 1 0 0	3	1 0 1 0 1 0 1	4
1 0 1	1 0 1 0	1 0 1 0 1 0 1	4	1 0 1 1 0 1 0	4
1 1 0	1 1 0 0	1 1 0 0 1 1 0	4	1 1 0 0 1 1 0	4
1 1 1	1 1 1 1	1 1 1 1 1 1 1	7	1 1 0 1 0 0 1	4

Tabelle 1.2: Codeworte der Verkettung von äußerem (4,3,2)-SPC und innerem (7,4,3)-Hamming-Code ($\tilde{\mathbf{c}}_2$ sind geschickt gewählte Codeworte zur Erhöhung der Mindestdistanz auf $d_{\min} = 4$)

Der äußere (4,3,2)-SPC-Code besitzt die Mindestdistanz $d_{\min}^{(1)} = 2$, während der innere (7,4,3)-Hammingcode

bekannterweise die Mindestdistanz $d_{\min}^{(2)} = 3$ hat. Sie stellt gemäß obiger Tabelle auch die kleinste Distanz des verketteten Codes dar. Hieraus folgt, dass die Verkettung zweier Codes nicht zwingend zu einem besseren Gesamtcode führt. Vielmehr wirkt sich der äußere Code gar nicht aus und die Mindestdistanz des inneren Codes bestimmt auch die Mindestdistanz des Gesamtcodes.

Verantwortlich für das Scheitern einer Codeverkettung ist die Zuordnung der Codeworte $\mathbf{c}_1$ des äußeren Codes auf die Codeworte $\mathbf{c}_2$ des inneren. Der äußere Codierer sorgt durch das Hinzufügen von Redundanz für eine Teilmengenbildung, d.h. den $2^4 = 16$ prinzipiell möglichen Eingangsworten am inneren Codierer treten tatsächlich nur $2^3 = 8$ auf. Dies hat zur Folge, dass von 16 möglichen Hamming-Codeworten nur 8 verwendet werden. Erfolgt die Auswahl der 8 verwendeten Codeworte so ungeschickt, dass sie zufällig die kleinst mögliche Hamming-Distanz untereinander aufweisen, wird hierdurch auch die Gesamtdistanz des verketteten Codes dominiert; der äußere Code wirkt sich nicht auf die Distanzeigenschaften aus. Hieraus kann das Ziel formuliert werden, dass die tatsächlich benutzten 8 Codeworte so zu wählen sind, dass sie möglichst große Distanzen untereinander aufweisen. Dann würde sich – wie in Tabelle 1.2 mit $\tilde{\mathbf{c}}_2$ gezeigt – für den Gesamtcode eine höhere minimale Hamming-Distanz (im Beispiel $d_{\min} = 4$) ergeben, als sie der innere Code besitzt.

Wie muss die Verkettung von Codes im allgemeinen erfolgen, damit sich für den verketteten Code optimale Distanzeigenschaften ergeben?

1.3.2 Produktcodes

Eine geschickte Verkettung von Codes stellen die Produktcodes dar. Auch wenn es auf den ersten Blick nicht sofort ersichtlich ist, stellt der folgende Aufbau die serielle Verkettung zweier Codes dar.

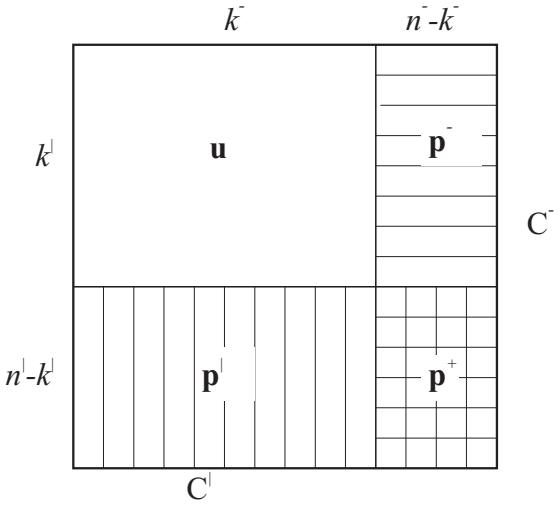

Bild 1.5: Allgemeiner Aufbau eines Produktcodes

Ein Matrix $\mathbf{u}$ bestehend aus $k^| \cdot k^-$ Informationsbit wird in $k^|$ Zeilen und k^- Spalten angeordnet. Die Informationsbit dieser Matrix werden dann zeilenweise einer *horizontalen Codierung* durch einen systematischen (n^-, k^-, d^-)-Blockcode C^- unterzogen. Jede Zeile stellt ein eigenes Codewort dar, so dass insgesamt $k^|$ Codewörter generiert wurden. Die erzeugten Prüfbit sind in der Prüfmatrix $\mathbf{p}^-$ zusammengefaßt. Die Coderate des horizontalen Codes beträgt $R_c^- = \frac{k^-}{n^-}$.

Anschließend erfolgt eine *vertikale Codierung* aller bisher erzeugten Codeworte mit einem ebenfalls systematischen $(n^|, k^|, d^|)$-Blockcode $C^|$, wobei jede Spalte ein eigenes Codewort darstellt. Die Prüfbit der Informationsmatrix $\mathbf{u}$ sind in $\mathbf{p}^|$ zusammengefaßt worden, die Prüfbit der Prüfmatrix $\mathbf{p}^-$ in $\mathbf{p}^+$. Die Coderate beträgt hier $R_c^| = \frac{k^|}{n^|}$.

Anmerkung:
Werden lineare Teilcodes eingesetzt (wir gehen in der Vorlesung immer davon aus), so stellen alle Zeilen als auch alle Spalten gültige Codeworte von C^- bzw. $C^|$ dar. Dies ist nicht unbedingt selbstverständlich, da die Prüfmatrix $\mathbf{p}^+$ lediglich durch die vertikale Codierung erzeugt wurde. Diese stellt aber eine Linearkombination der horizontalen Codeworte dar, wodurch sich aufgrund der Linearität wiederum ein gültiges Codewort aus C^- ergibt.

Die in Bild 1.5 dargestellte Anordnung entspricht einer seriellen Verkettung von C^- und $C^|$, da alle von C^- erzeugten Codeworte anschließend komplett durch $C^|$ abermals codiert werden.

Die Coderate lautet somit

$$R_c = \frac{k^- \cdot k^|}{n^- \cdot n^|} = R_c^- \cdot R_c^| \tag{1.2}$$

und setzt sich somit aus dem Produkt der Coderaten von C^- und $C^|$ zusammen. Für die Minimaldistanz des Produktcodes gilt

$$d_{\min} = d_{\min}^- \cdot d_{\min}^| \, . \tag{1.3}$$

Während Gl. (1.2) selbsterklärend ist, lässt sich Gl. (1.3) folgendermaßen veranschaulichen. Wir stellen uns vor, dass $\mathbf{u}$ nur eine Zeile enthält, die Binärstellen ungleich Null enthält. Das sich ergebende Codewort dieser Zeile habe das minimale Gewicht $d_{\min}^-$ des horizontalen Codes. Für jede der $d_{\min}^-$ Binärstellen ungleich Null (und nur für diese) erzeugt der vertikale Code $C^|$ genau $d_{\min}^-$ Codeworte, welche jeweils mindestens das Gewicht $d_{\min}^|$ besitzen. Die kleinste vorkommende Distanz ergibt sich also aus dem Produkt der minimalen Distanzen der beiden Teilcodes und ist somit größer als diese.

Frage: Worin besteht der Unterschied zwischen den Produktcodes und der einfachen seriellen Verkettung?

Der Produktcode aus Bild 1.5 besitzt inhärent einen Blockinterleaver. Dies ist der Tatsache zu entnehmen, dass C^- mit der zeilenweisen Codierung andere Informationsbit zu einem Codewort zusammenfaßt als $C^|$ mit der vertikalen Codierung. Produktcodes enthalten also einen Blockinterleaver mit n^- Spalten und $k^|$ Zeilen. Hierdurch erhöht sich die Mindestdistanz des verketteten Codes gegenüber dem Beispiel aus Abschnitt 1.3.1.

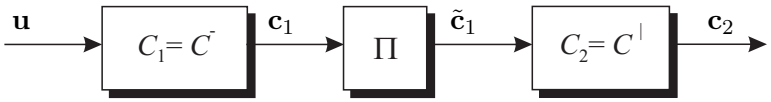

Bild 1.6: Serielle Verkettung zweier Codes mit Interleaving

Die Auswirkungen einer größeren Mindestdistanz sollen an folgendem Beispiel verdeutlicht werden.

Beispiel 3:
Wir konstruieren einen (12,6,4)-Produktcode aus den schon von Beispiel 1 in Abschnitt 1.3.1 bekannten (3,2,2)- und (4,3,2)-SPC-Codes. (siehe Bild 1.7). Die Coderate des Produktcodes $R_c = 6/12 = 1/2$ verändert sich nicht gegenüber der des verketteten Codes ohne Codespreizung. Der Unterschied besteht vielmehr darin, dass durch den Einsatz des Interleavers 3 Informationsworte $\mathbf{u}$ gemeinsam ein verkettetes Codewort ergeben. Hierdurch wurde die Blocklänge der einfachen Verknüpfung ohne Codespreizung von $n = 4$ auf jetzt $n = 12$ vergrößert, was – wie wir wissen – für Blockcodes ein großer Vorteil ist. Dies führt zu folgender Verbesserung.

Die als Teilcode verwendeten SPC-Codes besitzen beide die Mindestdistanz 2 und können somit 1 Fehler erkennen, allerdings keine Fehler korrigieren. Die Verknüpfung zu einem Produktcode erhöht nun nach Gl. (1.3) die Mindestdistanz auf 4, so dass 1 Fehler korrigiert und 3 Fehler erkannt werden können.

Die Fehlerkorrektur ist in Bild 1.7 noch einmal veranschaulicht. Ist das Symbol x_1 fehlerhaft, so sprechen sowohl die Zeilenprüfung der 2. Zeile als auch die Spaltenprüfung der 1. Spalte an. Beide können nur einen Fehler erkennen, ihr Schnittpunkt gibt jedoch exakt die fehlerhafte Stelle an.

x_0	x_4	x_8
x_1	x_5	x_9
x_2	x_6	x_{10}
x_3	x_7	x_{11}

Bild 1.7: Beispiel 3: (12,6,4)-Produktcode bestehend aus (3,2,2)- und (4,3,2)-SPC-Codes

Die Singleton-Schranke aus dem letzten Semester gab an, welche Mindestdistanz $d_{\min}$ ein (n,k)-Code maximal annehmen kann. Sie beträgt für das obige Beispiel

$$d_{\min} \leq n-k+1 = 12-6+1 = 7$$

und zeigt, dass dieser Produktcode auch kein MDS-Code (*Maximum Distance Separable*) ist (vgl. Hamming-Schranke: $2^{n-k} = 2^6 = 64 \geq \sum_{r=0}^{t} \binom{n}{r} = \sum_{r=0}^{1} \binom{12}{r} = 13$).

Beispiel 4:
Die beiden aus Beispiel 2 bekannten Teilcodes werden nun zu einem (28,12,4)-Produktcode kombiniert (siehe Bild 1.8). Die Coderate $R_c = 12/28 = 3/7$ verändert sich nicht gegenüber der des verketteten Codes ohne Codespreizung. Die neue minimale Gesamtdistanz beträgt jetzt $d_{\min} = 2 \cdot 3 = 6$. Damit lassen sich nun 2 Fehler korrigieren und 5 erkennen.

x_0	x_7	x_{14}	x_{21}
x_1	x_8	x_{15}	x_{22}
x_2	x_9	x_{16}	x_{23}
x_3	x_{10}	x_{17}	x_{24}
x_4	x_{11}	x_{18}	x_{25}
x_5	x_{12}	x_{19}	x_{26}
x_6	x_{13}	x_{20}	x_{27}

x_0	x_7	x_{14}	x_{21}
x_1	x_8	x_{15}	x_{22}
x_2	x_9	x_{16}	x_{23}
x_3	x_{10}	x_{17}	x_{24}
x_4	x_{11}	x_{18}	x_{25}
x_5	x_{12}	x_{19}	x_{26}
x_6	x_{13}	x_{20}	x_{27}

Bild 1.8: Beispiel 4: (28,12,6)-Produktcode bestehend aus (4,3,2)-SPC-Code und (7,4,3)-Hamming-Code

Die Singleton-Schranke lautet hier

$$d_{\min} \leq n-k+1 = 28-12+1 = 17$$

und zeigt, dass auch dieser Produktcode kein MDS-Code ist (Hamming-Schranke: $2^{14} \geq \sum_{r=0}^{2} \binom{29}{r} = 407$).

Bei diesen sehr einfachen und übersichtlichen Betrachtungen ist zu beachten, dass als Komponentencodes nicht sehr leistungsfähige Codes eingesetzt wurden. Zwar ist es das Ziel, mit einfachen Teilcodes mächtige verkettete Codes zu generieren, ein (3,2,2)-SPC-Code ist für seine Coderate von $R_c = 2/3$ allerdings nicht sehr geeignet. Wir werden später anhand einiger Beispiele sehen, dass sich Blockcodes insbesondere für hohe Coderaten eignen und gerade nicht für Raten von 1/2 oder geringer.

1.3.3 Wahl der Teilcodes

Selbstverständlich können nicht nur Blockcodes in der oben beschriebenen Art und Weise seriell miteinander verknüpft werden. Die gleiche Methode ist auch auf zwei oder mehrere Faltungscodes und auch auf Kombi-

nationen von Block- und Faltungscodes anwendbar. Prinzipiell stellt sich hier die Frage, welchen Code man als inneren und welchen man als äußeren Code wählt. Diese Frage ist nicht grundsätzlich zu beantworten, die Antwort hängt von vielen verschiedenen Faktoren ab. Beispielsweise ist es hinsichtlich des Distanzspektrums eines verketteten Codes von Vorteil, wenn der äußere Code eine möglichst große freie Distanz besitzt. Demnach müßte der stärkere von zwei Codes als äußerer Code eingesetzt werden.

Wie sich später allerdings zeigen wird, können verkettete Codes aus Gründen der Realisierbarkeit im allgemeinen nicht nach dem *Maximum Likelihood*-Kriterium decodiert werden. Vielmehr wird die in Abschnitt 1.7 beschriebene iterative Decodierung eingesetzt. Für sie ist es günstiger, den stärkeren Code innen einzusetzen, da dieser zuerst decodiert wird und somit eine bessere Ausgangsbasis für die Decodierung des äußeren Codes liefert.

Im Folgenden werden zwei serielle Codeverkettungen behandelt, die beide Faltungscodes einsetzen und die in Bild 1.6 dargestellte Struktur aufweisen. Die erste Verkettung besteht aus zwei über einen Interleaver verbundenen Faltungscodes. Der äußere Codierer kann problemlos terminiert werden, wohingegen der innere Codierer die codierte Sequenz des äußeren erhält und sein Trellisdiagramm somit offen bleibt.

Die zweite Verkettung setzt sich aus einem äußeren (terminierten) Faltungscode und einem inneren Walsh-Hadamard-Code zusammen. Der Walsh-Hadamard-Code ist ein linearer, systematischer Blockcode der Rate $\mathrm{ld}(M)/M$, d.h. er ordnet beispielsweise einem 6-Bit-Eingangswort ein 64-Bit-Ausgangswort zu. Damit ist der WH-Code ein niedrigratiger Code. Eine Besonderheit besteht darin, dass seine Ausgangsworte orthogonal zueinander sind. Aus diesem Grund werden WH-Codes auch als orthogonale Modulationsverfahren interpretiert. Die Kombination von äußerem Faltungscode und innerem WH-Code wird übrigens in einem Mobilfunksystem der zweiten Generation, dem Qualcomm-System IS-95 in den USA verwendet.

1.4 Parallele Codeverkettung (Turbo-Codes)

1.4.1 Modifikation der Produktcodes

Am Anfang dieses Kapitels wurde schon die parallele Verkettung von Codes angesprochen. Der Unterschied zur seriellen Verknüpfung besteht darin, dass hier **ausschließlich** die Informationsbit von mehreren Teilcodes codiert werden, die jeweiligen Prüfbit jedoch nicht. Dies lässt sich leicht auf die weiter vorne betrachteten Produktcodes übertragen. Wir können den in Bild 1.5 dargestellten Produktcode in eine parallele Verkettung zweier Codes überführen, indem wir die Prüfbit der Prüfbit $\mathbf{p}^+$ entfernen. Es ergibt sich dann folgende Struktur.

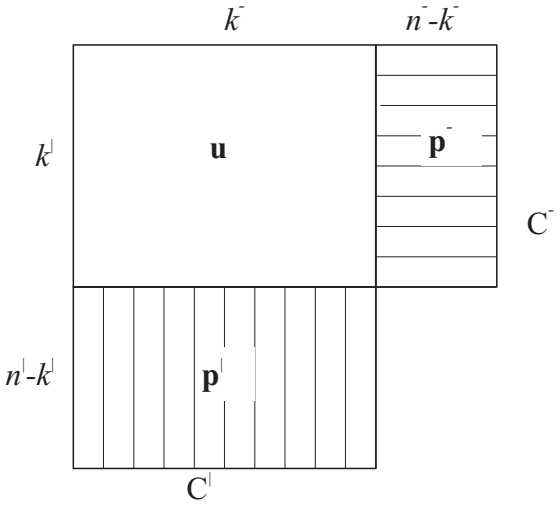

Bild 1.9: Allgemeine Struktur eines unvollständigen Produktcodes

Man spricht in diesem Zusammenhang auch von einem unvollständigen Produktcode. Er hat die Coderate

$$\begin{aligned} R_c &= \frac{k^- \cdot k^|}{k^- \cdot k^| + (n^- - k^-) \cdot k^| + (n^| - k^|) \cdot k^-} = \frac{1}{1 + (n^- - k^-)\frac{1}{k^-} + (n^| - k^|)\frac{1}{k^|}} \\ &= \frac{1}{1 + \frac{1}{R_c^-} - 1 + \frac{1}{R_c^|} - 1} = \frac{1}{\frac{1}{R_c^|} + \frac{1}{R_c^-} - 1} = \frac{R_c^| \cdot R_c^-}{R_c^- + R_c^| - R_c^- \cdot R_c^|} \end{aligned} \tag{1.4}$$

Hinsichtlich der Mindestdistanz gilt die Beziehung

$$d_{\min} = d_{\min}^- + d_{\min}^| - 1 \; . \tag{1.5}$$

Gl. (1.5) ist folgendermaßen zu verstehen. Wir nehmen wieder an, dass der Informationsteil $\mathbf{u}$ nur eine Zeile enthält, die exakt ein Element ungleich Null besitzt, alle übrigen Zeilen sollen nur Nullen enthalten. Ferner habe das zugehörige Codewort dieser Zeile das minimale Gewicht $d_{\min}^-$. Die vertikale Decodierung mit $C^|$ erzeugt somit nur ein einziges Codewort ungleich Null, da der Prüfteil nicht mehr codiert wird. Dieses Codewort hat aber ein minimales Gewicht von $d_{\min}^|$, so dass sich beide Gewichte addieren. Jetzt muss nur noch berücksichtigt werden, dass das Informationsbit, welches ungleich Null war, in beiden Codeworten vorkommt und nur einmal übertragen wird, woraus sich Gl. (1.5) ergibt.

Beispiel 5: Unvollständiger Produktcode mit (3,2,2)- und (4,3,2)-SPC-Codes
Wir wollen nun die beiden zuvor betrachteten Beispiele 3 und 4 auf unvollständige Produktcodes anwenden. Aus den beiden SPC-Codes konstruieren wir einen (11,6,3)-Produktcode (siehe Bild 1.10). Die Coderate $R_c = 6/11$ verändert sich in diesem Fall wenig gegenüber dem vollständigen Produktcode. Allerdings verringert sich die minimale Distanz gemäß Gl. (1.5) von 4 auf 3. Damit lässt sich immer noch 1 Fehler korrigieren, aber nur noch 2 Fehler erkennen.

x_0	x_4	x_8
x_1	x_5	x_9
x_2	x_6	x_{10}
x_3	x_7	

Bild 1.10: Beispiel 5: (11,6,4)-Produktcode bestehend aus (3,2,2)- und (4,3,2)-SPC-Codes

Die eingekreisten Elemente kennzeichnen die Binärstellen gleich 1 für ein mögliches Codewort mit minimalem Gewicht. Die Singleton-Schranke lautet

$$d_{\min} \leq n - k + 1 = 11 - 6 + 1 = 6.$$

Beispiel 6:
Die beiden aus Beispiel 4 bekannten Teilcodes werden nun zu einem unvollständigen (25,12,4)-Produktcode kombiniert (siehe Bild 1.11). Die Coderate $R_c = 12/25$ vergrößert sich etwas gegenüber dem vollständigen Produktcode. Die neue minimale Gesamtdistanz beträgt jetzt $d_{\min} = 2 + 3 - 1 = 4$. Damit lassen sich nun 2 Fehler erkennen und 1 Fehler korrigieren.

Die Singleton-Schranke lautet hier

$$d_{\min} \leq n - k + 1 = 25 - 12 + 1 = 14.$$

x_0	x_7	x_{14}	x_{21}
x_1	x_8	x_{15}	x_{22}
x_2	x_9	x_{16}	x_{23}
x_3	x_{10}	x_{17}	x_{24}
x_4	x_{11}	x_{18}	
x_5	x_{12}	x_{19}	
x_6	x_{13}	x_{20}	

Bild 1.11: Beispiel 6: (25,12,4)-Produktcode bestehend aus (4,3,2)-SPC-Code und (7,4,3)-Hamming-Code

1.4.2 Turbo-Codes

Die so genannten *Turbo-Codes* wurden erstmals im Jahr 1993 von Berrou, Glavieux und Thitimajshima vorgestellt. Ihre Leistungsfähigkeit versetzte die gesamte Fachwelt in helle Aufregung, war es doch erstmals gelungen, sich der Kapazitätsgrenze nach Shannon auf bis zu 0,5 dB zu nähern (gängige im letzten Semester vorgestellte Faltungscodes liegen etwa 3-5 dB von ihr entfernt). Dieser gewaltige Unterschied wird noch einmal in Bild 1.12 veranschaulicht. Es ist zu erkennen, dass mit zunehmender Einflusslänge L_c die Faltungscodes an Leistungsfähigkeit gewinnen, diese Gewinne scheinen aber immer kleiner zu werden. Damit ist abgesehen vom exponentiell steigenden Decodieraufwand mit dieser Maßnahme die Kapazitätsgrenze nach Shannon voraussichtlich nicht zu erreichen.

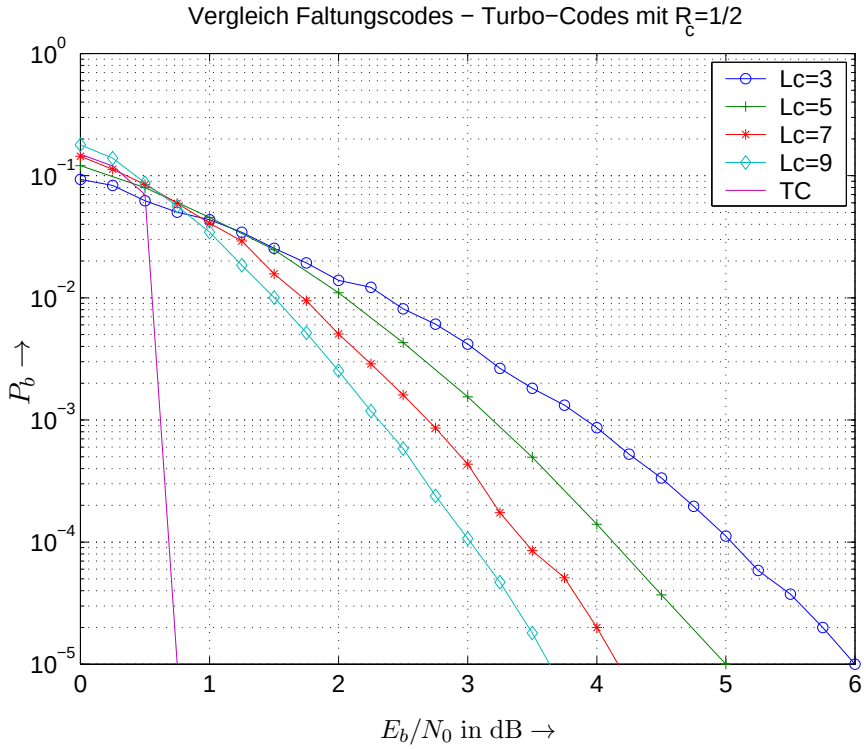

Bild 1.12: Vergleich der Bitfehlerraten für klassische Faltungscodes und Turbo-Code

Demgegenüber erreicht der hier dargestellte Turbo-Code eine Fehlerrate von $P_b = 10^{-5}$ bei 0,5 dB, was einem Gewinn gegenüber dem Faltungscode mit $L_c = 9$ von knapp 3 dB entspricht. Dies ist mit konventionellen Faltungscodes nicht zu erreichen.

Dieses Ergebnis ist ein wichtiger Meilenstein in der Codierungstheorie, da mit Hilfe der Turbo-Codes viele wichtige Eigenschaften verketteter Codes zu analysieren sind. Neben der geschickten Kombination mehrerer Codes, die Gegenstand dieses Abschnitts ist, spielt ferner der iterative Decodierprozeß eine wichtige Rolle. Seiner speziellen Struktur verdanken die Turbo-Codes auch ihren Namen. Da diese spezielle Art der Decodierung auch für die im vorigen Abschnitt behandelten Produktcodes angewendet wird, erfolgt die Beschreibung separat im nächsten Abschnitt.

Seit ihrer ersten Vorstellung beschäftigen sich weltweit unzählige Forscher mit den Turbo-Codes. Sie sorgten gewissermaßen für eine Initialzündung, den die iterative Decodierung wurde nun auf viele Bereiche auch außerhalb der Codierung angewandt (mehr dazu im nächsten Abschnitt). Einige sehr interessante und wichtige Eigenschaften der Turbo-Codes wurden erst nach ihrer Entwicklung analysiert und verstanden, so z.B. der eigentliche Grund für ihre bisher noch nie erreichte Leistungsfähigkeit. Wir wollen in dieser Vorlesung lediglich die Grundlagen der Turbo-Codes erläutern. Auch werden nicht alle Aussagen durch mathematische Beweise oder Herleitungen belegt. Vielmehr geht es darum, das prinzipielle Verständnis für diese spezielle Art der Codeverkettung zu verstehen.

Allgemeiner Aufbau von Turbo-Codes

Den generellen Aufbau eines Turbo-Codierers zeigt Bild 1.13. Sie bestehen aus einer parallelen Verkettung von i.a. q Faltungscodes, wobei die zum Einsatz kommenden Codes nicht zwingend unterschiedlich sein müssen. In den meisten Fällen werden sogar identische Teilcodes verwendet. Jeder Teilcodierer C_i erhält die gleiche Eingangsfolge $\mathbf{u}$, allerdings in jeweils unterschiedlicher Reihenfolge ($\mathbf{u}_i$). Die verschiedenen Permutationen von $\mathbf{u}$ in $\mathbf{u}_i$ werden über Interleaver Π_i realisiert, wobei der Interleaver Π_1 vor dem ersten Teilcode C_1 auch weggelassen werden kann.

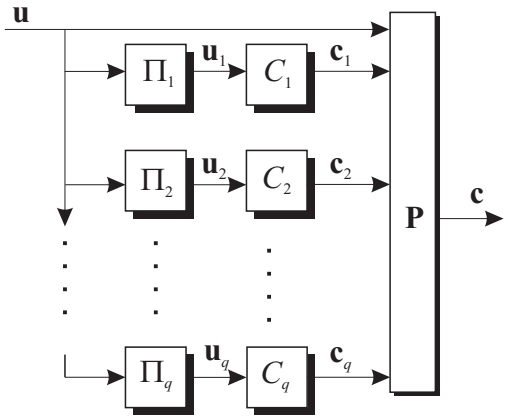

Bild 1.13: Allgemeiner Aufbau eines Turbo-Codes bestehend aus q Teilcodes

Als Teilcodes werden in der Regel systematische Codes eingesetzt, die Ausgänge der Codierer in Bild 1.13 liefern demnach nur die redundanten Prüfbit. Die Informationsbit $\mathbf{u}$ werden über den oberen Zweig direkt an den Ausgang geführt. **Turbo-Codes sind also systematische Codes!** Zur Anpassung der Gesamtcoderate können sowohl die Prüfbit als auch die Informationsbit punktiert werden. Hierzu dient die Punktierungsmatrix $\mathbf{P}$.

Die Coderate des verketteten Codes lässt sich wie folgt berechnen. Alle Codierer C_i erhalten Eingangssequenzen $\mathbf{u}_i$ gleicher Länge $k = L_\Pi$. Sie besitzen die Coderaten $R_{c,i} = k/n_i$. Da $\mathbf{u}$ nur einmal übertragen wird, fügt jeder Teilcode $n_i - k$ Prüfbit hinzu. Werden nur die Prüfbit der Teilcodierer C_i und nicht $\mathbf{u}$ punktiert, so ergibt sich insgesamt die Coderate

$$\begin{aligned} R_c &= \frac{k}{k + (n_1 - k) + (n_2 - k) + \cdots + (n_q - k)} = \frac{k}{\sum_{i=1}^{q} n_i - (q-1) \cdot k} \\ &= \frac{1}{\sum_{i=1}^{q} \frac{n_i}{k} - (q-1)} = \frac{1}{\sum_{i=1}^{q} \frac{1}{R_{c,i}} - (q-1)} \,. \end{aligned} \tag{1.6}$$

Aus Gründen der Übersichtlichkeit beschränken wir uns im Folgenden auf lediglich 2 Teilcodes. Mit dieser Einschränkung können weiterhin alle wesentlichen Aspekte erläutert werden. Eine Erweiterung auf mehr als 2 Komponentencodes ist allerdings jederzeit möglich (s. Bild 1.14). Die Gesamtcoderate für $q = 2$ lautet

$$R_c = \frac{1}{\frac{1}{R_{c,1}} + \frac{1}{R_{c,2}} - 1} = \frac{R_{c,1} \cdot R_{c,2}}{R_{c,1} + R_{c,2} - R_{c,1} \cdot R_{c,2}} \tag{1.7}$$

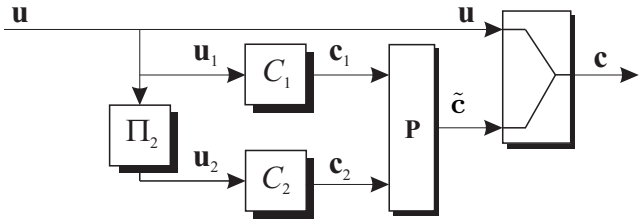

Bild 1.14: Aufbau eines Turbo-Codes bestehend aus 2 Teilcodes

1.4.3 Wahl der Teilcodes

Selbstverständlich hat die konkrete Wahl der Teilcodes auch bei der parallelen Verkettung direkte Auswirkungen auf die Leistungsfähigkeit des Gesamtcodes. Aus diesem Grund sollen hier nun einige wichtige Eigenschaften der Teilcodes beschrieben werden. Dabei beziehen sich die hier gemachten Aussagen auf die im letzten Abschnitt behandelten Turbo-Codes.

Als Teilcodes werden rekursive, systematische Faltungscodes (RSC-Code) eingesetzt. Daher erzeugen die Teilcodes C_i wie in Bild 1.15 dargestellt lediglich die Prüfbit, die Informationsbit werden am Ende gesondert hinzugefügt. Hinsichtlich der Einflusslänge L_c und der Coderate R_c der Teilcodes ist zu sagen, dass der Sinn der Codeverkettung die Synthese 'einfacher' Codes zu einem 'großen' Gesamtcode war. Aus diesem Grund wird L_c zwecks einfacher Decodierung nicht allzu groß gewählt und liegt in der Praxis im Bereich $3 \leq L_c \leq 5$. Selbstverständlich lassen sich Unterschiede in der Leistungsfähigkeit in Abhängigkeit von der Einflusslänge der Teilcodes feststellen. Die Coderate der C_i beträgt in der Praxis $R_c = 1/n$, da größere Raten immer durch Punktierung erreicht werden können. Es gibt aber auch Ansätze, Teilcodes höherer Rate einsetzen, die dann geringfügig bessere Ergebnisse erzielen.

Ein wesentlicher Grund für die herausragende Leistungsfähigkeit der Turbo-Codes ist die Rekursivität der Teilcodes. Man kann zeigen, dass Bitfehlerrate P_b und Interleavergröße L_Π über die Beziehung

$$P_b \sim L_\Pi^{1-w_{\min}} \tag{1.8}$$

zusammenhängen, wobei $w_{\min}$ das minimale Eingangsgewicht eines Teilcodes beschreibt, für das ein endliches Ausgangsgewicht erzielt wird. Bekannterweise besitzen RSC-Codes eine IIR-Struktur, so dass für ein endliches Hamminggewicht der Ausgangssequenz mindestens ein Eingangsgewicht von $w_{\min} = 2$ erforderlich ist. Daher gilt für RSC-Codes

$$P_b^{\mathrm{RSC}} \sim L_\Pi^{-1} \,, \tag{1.9}$$

d.h. die Bitfehlerrate ist proportional zum Kehrwert der Interleavergröße. Anders ausgedrückt, mit wachsendem L_Π wird der Turbo-Code immer besser!

Für NSC-Codes gilt hingegen $w_{\min} = 1$, da sie eine endliche Impulsantwort besitzen. Damit ist die Bitfehlerrate eines aus NSC-Codes zusammengesetzten Turbo-Codes entsprechend Gl. (1.8) nicht von der Interleavergröße abhängig. Somit ist klar, warum in der Praxis stets RSC-Codes zum Einsatz kommen.

Ferner kann gezeigt werden, dass die freie Distanz d_f kein geeigneter Parameter zur Optimierung der Teilcodes mehr ist. Vielmehr ist hier die sogenannte *effektive Distanz*

$$d_{\mathrm{eff}} = w_{\min} + 2 \cdot c_{\min} \tag{1.10}$$

von Vorteil. Dies ist folgendermaßen zu interpretieren.

Turbo-Codes sind systematische Codes, das Gewicht $w_{\min}$ der Infobit geht also auch in das Gesamtgewicht ein. Für $w_{\min}$ Einsen am Eingang sei das minimale Gewicht der Prüfbit eines Teilcodes $c_{\min}$. Bei 2 identischen Teilcodes beträgt dann das Mindestgewicht des Turbo-Codes für $w = 2$ gerade d_{eff}.

Hieraus folgt direkt, dass geeignete Teilcodes für $w = 2$ das Gewicht der Prüfbit maximieren müssen. Dieses Ziel kann erreicht werden, wenn das Polynom zur Rückkopplung teilerfremd (prim) ist. Dann nämlich bildet das Schieberegister eine Sequenz maximaler Länge (m-Sequenz), d.h. der minimale Abstand der beiden Einsen am Eingang wird unter der Nebenbedingung einer endlichen Ausgangssequenz maximal. Hierdurch werden mit $w = 2$ die längst möglichen Sequenzen erzeugt, die natürlich ein höheres Gewicht als kurze Sequenzen haben können. Die übrigen Generatoren der Teilcodes sind dann derart zu wählen, dass das Gewicht der Prüfbit für das spezielle teilerfremde Rückkopplungspolynom maximal wird!

Beispiel 7: Turbo-Code mit $q = 2$ identischen Teilcodes mit $L_c = 3$ und $R_c = 1/2$

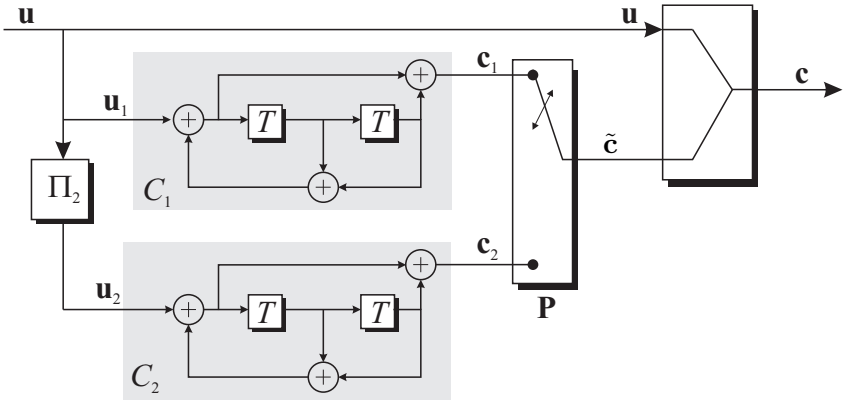

Bild 1.15: Beispiel 7: Struktur eines Turbo-Codes mit $R_c = 1/2$ bestehend aus 2 Teilcodes mit $\mathbf{g}_1 = 5_8$ und $\mathbf{g}_2 = 7_8$

- Prüfbit der Teilcodes werden alternierend übertragen $\longrightarrow \mathbf{P} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

  $\longrightarrow$ Coderate der Teilcodes ist $R_{c,1} = R_{c,2} = 2/3$, die des Gesamtcodes $R_c = 1/2$

- Einsatz rekursiver Teilcodes mit $\mathbf{g}_1 = 5$ und $\mathbf{g}_2 = 7$, wobei $\mathbf{g}_2$ zur Rückkopplung eingesetzt wird

- Polynom $g_2(D) = 1 + D + D^2$ ist teilerfremd

  $\longrightarrow$ Schieberegister bildet eine Sequenz maximaler Länge (m-Sequenz) mit $L = 2^2 - 1 = 3$

  $\longrightarrow\ d_{\text{eff}} = 2 + 2 \cdot 4 = 10$

- Polynom $g_1(D) = 1 + D^2$ ist **nicht** teilerfremd, denn es gilt $1 + D^2 = (1 + D)^2$

  $\longrightarrow$ Schieberegister bildet eine Sequenz mit der Länge $L = 2$

  $\longrightarrow\ d_{\text{eff}} = 2 + 2 \cdot 3 = 8$

$\longrightarrow$ Teilcode mit $\mathbf{g}_1$ als Rückkopplungspolynom würde schlechteren Turbo-Code bilden!

Beispiel 8: Turbo-Code mit $q = 2$ Teilcodes mit $L_c = 5$ und $R_c = 2/3$

- Teilcodes der Originalrate $R_c = 1/2$ werden zur Rate $R_c = 4/5$ punktiert

  $\longrightarrow \mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

  $\longrightarrow R_c = 2/3$

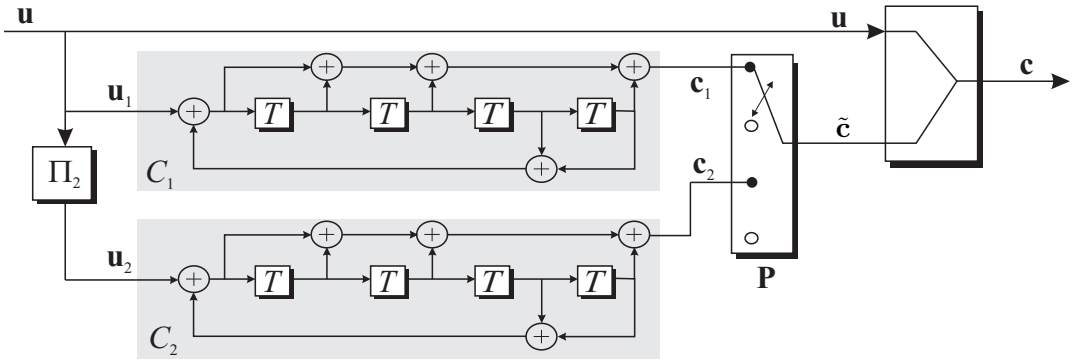

Bild 1.16: Beispiel 8: Struktur eines Turbo-Codes mit $R_c = 2/3$ bestehend aus 2 Teilcodes mit $\mathbf{g}_1 = 23_8$ und $\mathbf{g}_2 = 35_8$

- Einsatz rekursiver Teilcodes mit $\mathbf{g}_1 = 23_8$ und $\mathbf{g}_2 = 35_8$, wobei $\mathbf{g}_1$ zur Rückkopplung eingesetzt wird
- Polynom $g_1(D) = 1 + D^3 + D^4$ ist teilerfremd
  $\longrightarrow$ Schieberegister bildet eine Sequenz maximaler Länge (m-Sequenz) mit $L = 2^4 - 1 = 15$
- Polynom $g_2(D) = 1 + D + D^2 + D^4$ ist **nicht** teilerfremd, denn es gilt $D^4 + D^2 + D + 1 = (D + 1) \cdot (D^3 + D^2 + 1)$
  $\longrightarrow$ Schieberegister erzeugt periodische Sequenzen der Längen $L = 1$ und $L = 7$

$\longrightarrow$ Teilcode mit $\mathbf{g}_2$ als Rückkopplungspolynom würde schlechteren Turbo-Code bilden

Hinsichtlich der Punktierung der Teilcodes ist darauf zu achten, dass keiner der beteiligten Komponentencodes katastrophal wird (s. 1. Semester). Mit zunehmender Punktierung sind in der Regel Teilcodes größerer Einflusslänge zu wählen, damit die kleinste Distanz größer als eins bleibt und somit ein Codiergewinn erhalten bleibt.

1.5 Einfluss der Interleavers

Eine ebenso wichtige Rolle wie die Teilcodes spielt sowohl für die serielle wie auch für die parallele Verkettung der Interleaver. Er soll vermeiden, dass der Gesamtcode Ausgangsfolgen mit geringem Gewicht enthält. Für parallele Verkettungen wie den Turbo-Codes soll der Interleaver verhindern, dass an beiden Teilcodierern gleichzeitig Sequenzen $\mathbf{c}_i$ mit geringem Gewicht der Prüfbit auftreten. Dann hätte nämlich die Ausgangssequenz $\mathbf{c}$ des gesamten Turbo-Codes ein geringes Gewicht und der Code aufgrund der Linearität eine geringe kleinste Distanz. Dies muss vermieden werden, d.h. wenn die Ausgangsbit von C_1 ein geringes Gewicht haben, sollte der Interleaver Π_2 die Eingangsfolge $\mathbf{u}$ derart permutieren, dass die zu $\mathbf{u}_2$ gehörende Ausgangsfolge $\mathbf{c}_2$ ein hohes Gewicht besitzt. Dann hat der Turbo-Code insgesamt eine höhere Mindestdistanz. Das Verwürfeln der Eingangssequenz $\mathbf{u}$ führt also nicht nur zu einer Permutation der codierten Sequenz von $\mathbf{c}_1$ nach $\mathbf{c}_2$, sondern sorgt dafür, dass C_2 eine völlig andere Ausgangssequenz generiert. **Der Interleaver beeinflußt also direkt die Mindestdistanz des Turbo-Codes.**

Ähnliches gilt auch für die serielle Verkettung. Bekanntlich sorgt der äußere Code für eine Auswahl bestimmter Codeworte bzw. Codesequenzen des inneren Codes. Der Interleaver hat hier die Aufgabe, die Teilmenge des inneren Codes so auszuwählen, dass die Distanzeigenschaften des Teilcodes optimiert werden. Werden beispielsweise ein Faltungscode als äußerer Code und ein linearer Blockcode als innerer Code eingesetzt, so

sind die 'Einsen' der faltungscodierten Sequenz derart zu verschachteln, dass sie auf möglichst viele verschiedene Codeworte des Blockcodes aufgeteilt werden. Dann nämlich besitzt der Gesamtcode eine optimale freie Distanz.

Ein weiterer wichtiger Aspekt, der im nächsten Abschnitt auch noch diskutiert wird, betrifft nicht das Mindestgewicht selbst, sondern die Anzahl der Pfade mit einem bestimmten Gewicht. Durch das Interleaving wird nämlich die Anzahl der Sequenzen mit geringem Gewicht drastisch reduziert. Wie wir wissen, geht diese in die Koeffizienten c_d des Distanzspektrums und somit auch direkt in die Bitfehlerrate ein. **Der Interleaver beeinflußt also auch die Häufigkeit von Sequenzen mit bestimmten Gewicht und somit das gesamte Distanzspektrum.**

Ein letzter wichtiger Gesichtspunkt betrifft die statistische Unabhängigkeit der $\mathbf{c}_i$ an den Ausgängen der einzelnen Teilcodierer und greift somit den schon oben diskutierten Punkt wieder auf. Durch die Permutation der Informationssequenz erzeugen beide Teilcodierer unterschiedliche Ausgangssequenzen. Für den Decodierprozeß ist es sehr wichtig, dass $\mathbf{c}_1$ und $\mathbf{c}_2$ möglichst statistisch unabhängig sind (s. Abschnitt 1.7). Dies muss durch den Interleaver gewährleistet werden. Es ist leicht einzusehen, dass diese Forderung um so besser erfüllt wird, je größer der Interleaver ist, da dann die einzelnen Binärstellen weiter auseinander gespreizt werden können.

Während bei den Produktcodes vorwiegend einfache Blockinterleaver eingesetzt werden, finden bei den Turbo-Codes ab einer bestimmten Blocklänge quasi-zufällige Interleaver Verwendung. Der Grund hierfür kann wie folgt erklärt werden. Wir nehmen an, dass Teilcodes eingesetzt werden, deren rekursives Polynom zu einer m-Sequenz mit Periodendauer L führt. Zwei Einsen im Abstand L zueinander würden dann zu einer Ausgangssequenz mit endlichem Gewicht führen. Entsprechend der obigen Diskussion wäre es jetzt wünschenswert, wenn durch die Permutation Π_2 eine Ausgangssequenz generiert wird, die ein wesentlich größeres Gewicht besitzt. Dies wird bei Blockinterleavern für folgende Konstellation aber gerade nicht erreicht.

	$\vdots$		$\vdots$	
$\cdots$	1	$\cdots$	1	$\cdots$
	$\vdots$		$\vdots$	
$\cdots$	1	$\cdots$	1	$\cdots$
	$\vdots$		$\vdots$	

Sind vier Einsen quadratisch mit dem Abstand L zu den direkten Nachbarn angeordnet, so bleibt nach der Permutation mit einem Blockinterleaver diese Anordnung erhalten. Dies führt dazu, dass beide Teilcodes bei bestimmten Eingangssequenzen Codefolgen mit geringem Gewicht erzeugen. Einen Ausweg bieten *Random Interleaver*, die eine quasi zufällige Permutation durchführen und somit diese regelmäßigen Strukturen aufbrechen. Der Fall, dass beide Teilcodes niedergewichtige Sequenzen erzeugen, tritt dann wesentlich seltener auf.

Beispiel: Teilcodes aus Beispiel 7 mit $L_c = 3$, Blockinterleaver mit Länge 4x4=16
Originalinformationsfolge: $\mathbf{u} = (1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0)$
Informationsfolge nach Interleaving: $\mathbf{u}_2 = (1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0)$

1	0	1	0
0	0	0	0
1	0	1	0
0	0	0	0

Hieraus folgt, dass die obige Eingangsfolge $\mathbf{u}$ aus dem Interleaving unverändert hervorgeht und somit beide Komponentencodes C_1 und C_2 gleiche Ausgangssequenzen $\mathbf{c}_1$ bzw. $\mathbf{c}_2$ erzeugen. Zur Vermeidung solcher

Effekte müssen regelmäßige Strukturen im Interleaver vermieden werden. Da die Codespreizung nicht rein zufällig erfolgen kann, werden häufig durch systematische Rechnersuche optimierte Interleaver *gezüchtet*. Dazu geht man von einer im Prinzip willkürlich gewählten Startpermutation aus und betrachtet zunächst alle Eingangsfolgen mit einem Gewicht von $w_H(\mathbf{u}) = 2$. Die Permutationsvorschrift wird für diese Sequenzen nun so geformt, dass der obige Effekt nicht mehr auftritt. Anschließend wird die Prozedur von Eingangssequenzen mit $w_H(\mathbf{u}) = 3$ wiederholt usw. Natürlich ist der Rechenaufwand extrem hoch, weshalb diese Optimierungsstrategie aus Gründen der Realisierbarkeit nur für geringe Gewichte durchgeführt werden kann und somit auch nicht zu einem optimalen Interleaver führt. Die Verbesserungen gegenüber der Ausgangspermutation sind allerdings beachtlich.

1.6 Distanzeigenschaften und Abschätzung der Leistungsfähigkeit

Wie in den vorangegangenen Abschnitten deutlich wurde, haben Teilcodes als auch Interleaver entscheidenden Einfluss auf das Distanzspektrum und somit auch die Leistungsfähigkeit verketteter Codes. Daher soll in diesem Abschnitt das Distanzspektrum von verketteten Codes in sehr kurzer Form erläutert und damit die Frage beantwortet werden, warum verkettete Codes und im speziellen die Turbo-Codes eigentlich so gut sind. Anhand der uns schon bekannten Abschätzung

$$P_b \leq \frac{1}{2} \cdot \sum_{d=d_f}^{\infty} c_d \cdot \operatorname{erfc}\left(\sqrt{\frac{dR_c E_b}{N_0}}\right) \tag{1.11}$$

können vorab schon zwei prinzipielle Aussagen getroffen werden.

1. Die freie Distanz d_f eines Codes, d.h. die kleinste Hammingdistanz, die zwei Codesequenzen zueinander haben können, sollte so groß wie möglich sein.
2. Die Anzahl der Pfade insbesondere mit kleinen Distanzen sollte so klein wie möglich sein.

Um den immens hohen Aufwand der Berücksichtigung eines konkreten Interleavers zu umgehen, bedienen wir uns eines theoretischen Hilfsmittels, dem sogenannten *Uniform Interleaver*. Er berücksichtigt alle möglichen Permutationsvorschriften eines Interleavers der Länge L_π und repräsentiert somit einen 'mittleren' Interleaver.

Aus dem letzten Semester ist uns noch die IOWEF (*Input Output Weight Enumerating Function*)

$$A(W, D) = \sum_{w=0}^{k} \sum_{d=0}^{n} A_{w,d} \cdot W^w D^d$$

bekannt, welche auch öfters als Distanzspektrum bezeichnet wurde. Sie ist für verkettete Codes leicht zu modifizieren. So lautet beispielsweise die bedingte IOWEF für ein bestimmtes Eingangsgewicht w

$$A(w, D) = \sum_{d=0}^{n} A_{w,d} \cdot D^d \tag{1.12}$$

und für ein bestimmtes Ausgangsgewicht d

$$A(W, d) = \sum_{w=0}^{k} A_{w,d} \cdot W^w \ . \tag{1.13}$$

Wir müssen nun zwischen der seriellen und der parallelen Verkettung unterscheiden.

Parallele Verkettung

Eine wichtige Eigenschaft der parallelen Verkettung ist die Tatsache, dass alle Teilcodierer C_i jeweils eine permutierte Version $\mathbf{u}_i$ der gleichen Eingangssequenz $\mathbf{u}$ erhalten. Aus diesem Grund besitzen alle $\mathbf{u}_i$ das gleiche Eingangsgewicht w. Weiterhin erzeugen die Teilcodes nur die Prüfbit, während die Informationsbit direkt an den Ausgang gelangen. Aus diesem Grund wird statt des Gewichts d der gesamten codierten Sequenz nur das Gewicht c der Prüfbit in der IOWEF der Teilcodes berücksichtigt (es gilt $d = w + c$). Unter Verwendung des *Uniform Interleaver* lautet dann die bedingte IOWEF des parallel verketteten Codes für 2 Teilcodes

$$A^{\mathrm{par}}(w, C) = \frac{A_1(w, C) \cdot A_2(w, C)}{\binom{L_\pi}{w}} = \sum_c A^{\mathrm{par}}_{w,c} \cdot C^c \,. \tag{1.14}$$

Durch die Multiplikation der bzgl. w bedingten IOWEF's $A_1(w, D)$ und $A_2(w, D)$ der Teilcodes C_1 und C_2 wird erreicht, dass stets zwei Ausgangssequenzen mit gleichem Eingangsgewicht kombiniert werden. Außerdem bewirkt diese Multiplikation die Erfassung aller Kombinationsmöglichkeiten der Ausgangssequenzen beider Teilcodes (*Uniform Interleaver*). Die Division sorgt für die notwendige Mittelung, da $\binom{L_\pi}{w}$ die Anzahl der Permutationsmöglichkeiten von w Einsen in einem Block der Länge L_π angibt. Die Koeffizienten c_d aus Gl. (1.11) ergeben sich zu

$$c_d = \sum_{w+c=d} \frac{w}{L_\pi} \cdot A^{\mathrm{par}}_{w,c} \,. \tag{1.15}$$

Serielle Verkettung

Der wichtigste Unterschied zur parallelen Verkettung besteht bei der seriellen Verkettung darin, dass das Ausgangsgewicht des äußeren Codes gleich dem Eingangsgewicht des inneren Codes ist. Wir erhalten somit die IOWEF des Gesamtcodes

$$A^{\mathrm{ser}}(W, D) = \sum_l \frac{A_1(W, l) \cdot A_2(l, D)}{\binom{L_\pi}{l}} = \sum_w \sum_d A^{\mathrm{ser}}_{w,d} \cdot W^w D^d \,. \tag{1.16}$$

Außerdem ist zu beachten, dass der Interleaver nicht mit den Informationsbit $\mathbf{u}$ gefüllt wird, sondern mit der codierten Sequenz des äußeren Codes $\mathbf{c}_1$. Deshalb erfolgt die Mittelung über den Faktor $\binom{L_\pi}{l}$. Für die c_d gilt

$$c_d = \sum_w \frac{w}{L_\pi \cdot R_c^1} \cdot A^{ser}_{w,d} \,. \tag{1.17}$$

Bild 1.17 stellt die Koeffizienten c_d über der Distanz d für den Turbo-Code aus Beispiel 7 mit zwei identischen Komponentencodes $\mathbf{g}_1 = 7_8$ und $\mathbf{g}_2 = 5_8$ (Gesamtrate $R_c = 1/3$) und den Interleaverlängen $L_\Pi = 100$ und $L_\Pi = 400$ dar. Zum Vergleich sind die c_d auch für einen drittelratigen Faltungscode der Einflusslänge $L_c = 9$ illustriert. Wegen der Verwendung des *Uniform Interleavers* nehmen die c_d auch Werte kleiner Eins an, sie stellen Mittelwerte über alle möglichen Permutationsvorschriften dar.

- Der Turbo-Code besitzt mit $d_f = 5$ eine deutlich kleinere freie Distanz als der Faltungscode ($d_f = 18$)
- Die Koeffizienten c_d der Turbo-Codes sind aufgrund des Interleavings allerdings deutlich kleiner als die des Faltungscodes
- Der Unterschied nimmt mit wachsender Interleavergröße zu (Beachte logarithmische Darstellung!)

Bild 1.18 zeigt die mit den obigen Koeffizienten und Gl. (1.11) berechneten Bitfehlerkurven (*Union Bound*) für den AWGN- und auch den 1-Pfad Rayleigh-Fading-Kanal. Für beide Kanäle ergeben sich in etwa die gleichen Verhältnisse, weshalb im Folgenden nur auf den AWGN-Kanal näher eingegangen wird. Folgende Effekte sind zu beobachten:

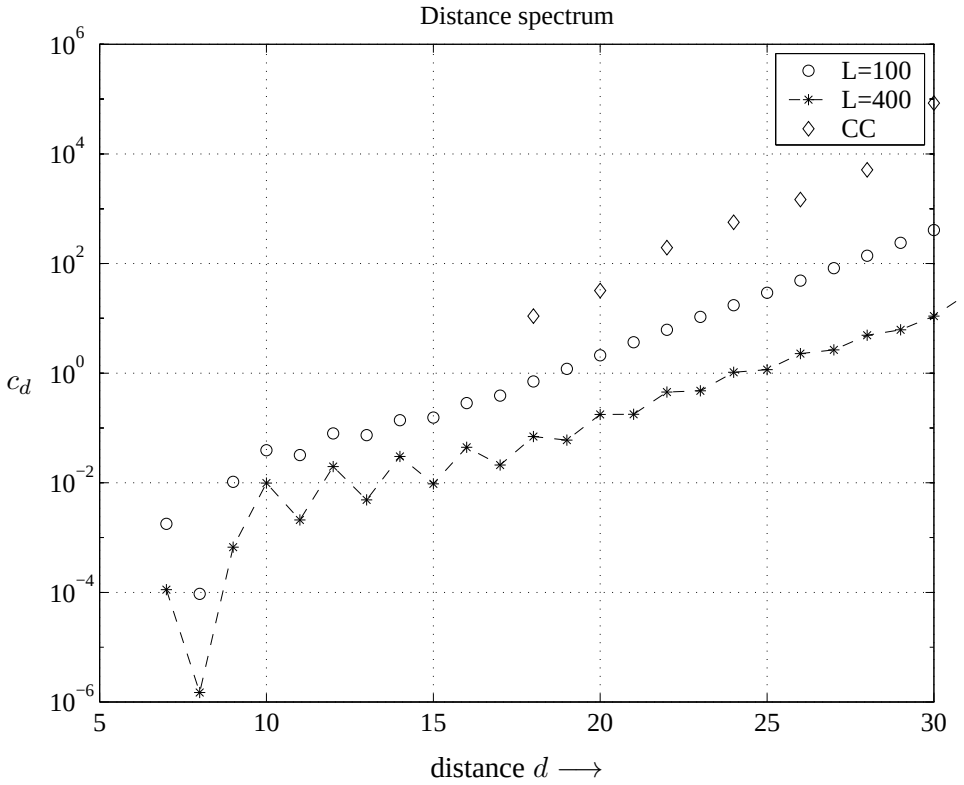

Bild 1.17: Distanzspektrum von Turbo-Code aus Beispiel 7 für verschiedene Interleavergrößen

- Für kleine Signal-Rausch-Abstände sind die Turbo-Codes wesentlich besser als die Faltungscodes (abgesehen vom Divergenzbereich der *Union Bound*)
- Die Vorteile werden mit steigender Interleavergröße größer
- Im weiteren Verlauf flachen die Bitfehlerkurven der Turbo-Codes ab
- Die des Faltungscodes werden dagegen immer steiler, so dass sich die Kurven zwischen 4 dB und 6 dB je nach Interleavergröße schneiden und die Faltungscodes besser sind

Erklärungen:

- Die freie Distanz bestimmt den asymptotischen Verlauf der Bitfehlerkurve (Verlauf für sehr große Signal-Rausch-Abstände). Daher ist es nicht verwunderlich, wenn in diesen Bereichen der Faltungscodes den Turbo-Codes überlegen ist. Es ist allerdings darauf zu achten, dass die Fehlerrate für diese Signal-Rausch-Abstände schon sehr niedrig ist und viele praktische Systeme für höhere Fehlerraten ausgelegt sind (z.B. $P_b = 10^{-3}$).
- Im Bereich kleiner Signal-Rausch-Abstände scheint die freie Distanz nur eine untergeordnete Rolle zu spielen, sonst wären hier die Turbo-Codes nicht so leistungsfähig. Vielmehr kommt es hier auf die Anzahl von Sequenzen mit bestimmtem Gewicht an.
  - Pfade mit geringem Gewicht werden hier fast ständig verwechselt; sie sollten - wenn überhaupt - nur sehr selten auftreten (kleines c_d). Eigentlich ein Vorteil für Faltungscode, der erst ab $d = 18$ Sequenzen besitzt.
  - Mit steigender Distanz wachsen die c_d bei Faltungscodes stark an; man kann zeigen, dass sie die Bitfehlerrate für kleine Signal-Rausch-Abstände dominieren (c_d wachsen schneller als die $\mathrm{erfc}()$ abfällt).
  - Für Turbo-Codes wachsen die c_d mit steigender Distanz wesentlich langsamer, wodurch Sequenzen mit kleinen Distanzen die Bitfehlerrate dominieren.

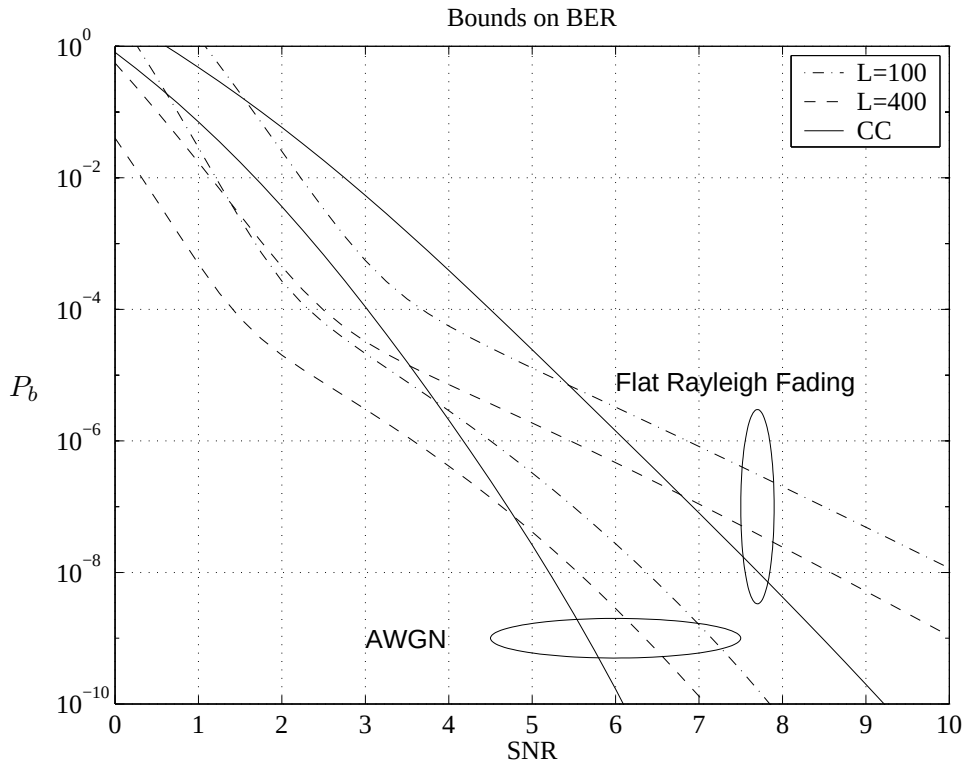

Bild 1.18: Analytische Abschätzung der Bitfehlerrate für Turbo-Codes aus Beispiel 7

Aus den obigen Erläuterungen folgt, dass zum Erreichen der Kapazitätsgrenze nach Shannon nicht nur die freie Distanz d_f, sondern auch die Anzahl der Pfade mit bestimmtem Gewicht ausschlaggebend ist. Hieraus ergeben sich andere Optimierungskonzepte als von den Faltungs- und Blockcodes bisher bekannt war.

1.7 Decodierung verketteter Codes

Eine sehr große Bedeutung kommt der Decodierung von verketteten Codes zu. Die analytische Abschätzung der Bitfehlerwahrscheinlichkeit mit Hilfe der *Union Bound* basiert stets auf der Annahme einer optimalen *Maximum Likelihood*-Decodierung. Diese ist aber gerade für verkettete Systeme i.a. nicht durchführbar. Zweck der Codeverkettung war es ja auch, relativ einfache Teilcodes zu verwenden, die sich mit geringem Aufwand decodieren lassen.

Wir erhalten somit auch im Empfänger eine Verkettung mehrerer Decodierer. Diese Konstellation führt unweigerlich zu einer suboptimalen Decodierung, die nicht mehr das *Maximum-Likelihood*-Kriterium erfüllt. Außerdem ist zu beachten, das bisher sowohl für Faltungscodes als auch für Blockcodes nur *Hard-Decision*-Algorithmen betrachtet wurden, d.h. Decodierverfahren, die als Ergebnis hart entschiedene Bit ausgeben. Während der für Faltungscodes verwendete Viterbi-Algorithmus zumindest in der Lage ist, *Soft*-Werte des Kanals zu verarbeiten, setzen die gängigen Verfahren für Blockcodes normalerweise eine *Hard-Decision* vor der Decodierung voraus. Jede harte Entscheidung vor der letzten Empfängerstufe resultiert aber automatisch in einem Informationsverlust, der nicht mehr kompensiert werden kann.

Um mit der verketteten Decodierung so dicht wie möglich an die optimale *Maximum-Likelihood*-Decodierung zu gelangen, sind also Verfahren erforderlich, die sowohl *Soft*-Werte verarbeiten als auch ausgeben können. Diese werden *Soft-In/Soft-Out*-Algorithmen genannt und spielen in der aktuellen Forschung eine wichtige Rolle. Wir wollen hier zuerst ein geeignetes Maß für die *Soft*-Information einführen und dann einige wichtige Algorithmen kennenlernen, die diese Information bereitstellen können.

1.7.1 Definition der *Soft*-Information

Im letzten Semester haben wir zwei verschiedene Decodierprinzipien kennengelernt, das MAP-Kriterium (Maximum a posteriori) und das *Maximum-Likelihood*-Kriterium. Der Unterschied besteht darin, dass bei letzterem alle Eingangssignale als gleichwahrscheinlich angenommen werden, während beim MAP-Kriterium eine unterschiedliche Verteilung des Eingangsalphabets berücksichtigt wird. Da dies für die weitere Betrachtung sehr wichtig ist, wollen wir im Folgenden stets das MAP-Kriterium verwenden, wobei die Kanalcodierung bei der Vorstellung der *Soft*-Information zunächst vernachlässigt wird. Die bisherige Notation wird hier weitgehend übernommen, so dass die logischen Bit $u\in\{0,1\}$ per BPSK-Modulation den Kanalsymbolen entsprechend

$$u = 0 \rightarrow x = +1 \tag{1.18}$$

bzw.

$$u = 1 \rightarrow x = -1 \tag{1.19}$$

zugeordnet werden. Das MAP-Kriterium lautet damit

$$P(u=0|y) = P(x=+1|y) \underset{>}{\overset{<}{}} P(u=1|y) = P(x=-1|y) \tag{1.20}$$

Mit Hilfe der Bayes-Regel können wir Gl. (1.20) umformen, wobei $P(\cdot)$ die Auftrittswahrscheinlichkeit eines Symbols beschreibt, während $p(\cdot)$ die Wahrscheinlichkeitsdichte angibt. Wir erhalten

$$\begin{aligned} & \frac{p(x=+1,y)}{p(y)} \underset{>}{\overset{<}{}} \frac{p(x=-1,y)}{p(y)} \\ \Longleftrightarrow \quad & \frac{p(x=+1,y)}{p(x=-1,y)} \underset{>}{\overset{<}{}} 1 \\ L(\hat{x}) \quad := \quad & \ln\frac{p(x=+1,y)}{p(x=-1,y)} = \underbrace{\ln\frac{p(y|x=+1)}{p(y|x=-1)}}_{L(y|x)} + \underbrace{\ln\frac{P(x=+1)}{P(x=-1)}}_{L_a(x)} \underset{>}{\overset{<}{}} 0\ . \end{aligned} \tag{1.21}$$

Die L-Werte in Gl. (1.21) werden *log-likelihood-ratios* genannt, da sie aus dem Logarithmus eines Wahrscheinlichkeitsverhältnisses hervorgehen. Sie stellen die sogenannte *Soft*-Information dar, da ihr Vorzeichen eine harte Entscheidung über das betrachtete Bit x angibt, während ihr Betrag ein Maß für die Zuverlässigkeit dieser Entscheidung darstellt.

Das logarithmische Verhältnis der Wahrscheinlichkeiten $\ln\frac{P(u=0)}{P(u=1)}$ (*log-likelihood ratio*) ist ein geeignetes Maß für die Zuverlässigkeit einer Entscheidung.

Dies heißt jedoch nicht, dass das in Gl. (1.21) dargestellte *Log-Likelihood*-Verhältnis die optimale Zuverlässigkeitsinformation ist. Sie erfüllt lediglich die rein anschaulich motivierte Vorstellung einer Zuverlässigkeitsinformation, was allerdings auch auf die Vorschrift

$$L(x) = P(x=+1) - P(x=-1)$$

zutrifft. Auch hier nimmt L positive Werte für $P(x=+1) > P(x=-1)$ und ansonsten negative Werte an. Trotzdem hat sich in der Praxis das LLR als Zuverlässigkeitsinformation durchgesetzt, weshalb wir es auch im Rahmen dieser Vorlesung verwenden wollen.

Der geschätzte *Soft*-Wert $L(\hat{x})$ in Gl. (1.21) setzt sich bei einer uncodierten Übertragung aus zwei Anteilen zusammen, dem Term $L(y|x)$, der die Übergangswahrscheinlichkeiten des Kanals enthält, und dem Term $L_a(x)$, welcher unabhängig von den Kanalausgangswerten y ist und **a-priori-Wissen** über das gesendete Symbol x repräsentiert. Ist dem Empfänger beispielsweise bekannt, mit welcher Wahrscheinlichkeit $x=+1$ bzw. $x=-1$

auftreten, kann dies gewinnbringend genutzt werden. Aufgrund der statistischen Unabhängigkeit der a-priori-Information von den empfangenen Symbolen können die LLR's einfach addiert werden (dies gilt für statistisch abhängige Signale nicht). Da x ein rein binäres Signal darstellt, gilt selbstverständlich

$$P(x=+1)+P(x=-1)=1$$

und wir erhalten den in Bild 1.19 dargestellten Verlauf von $L_a(x)$ in Abhängigkeit von $P(x=1)$.

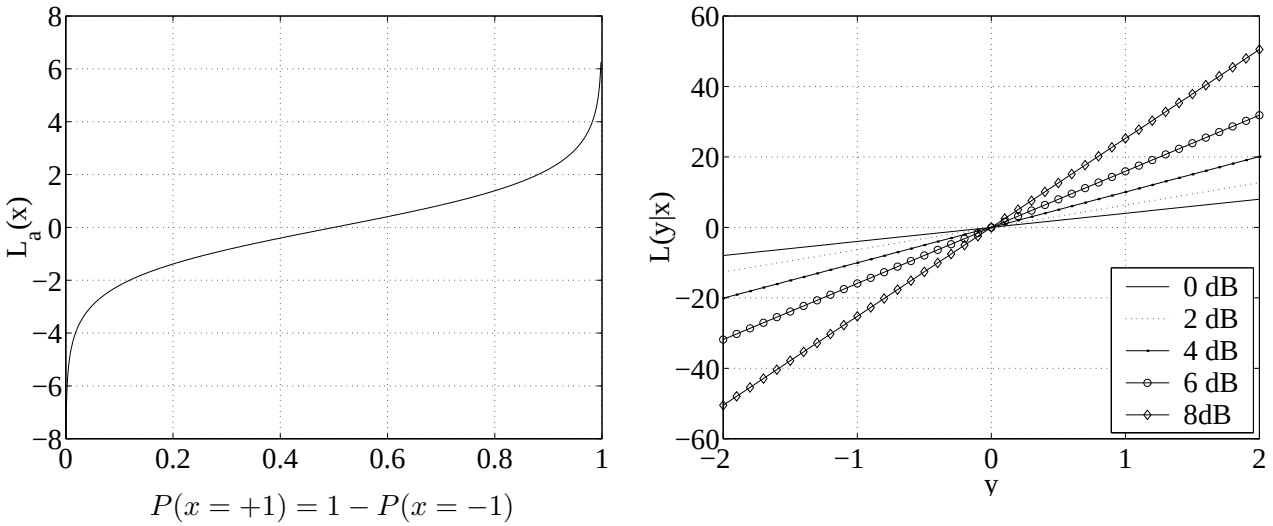

Bild 1.19: Verlauf des *Log-Likelihood*-Verhältnisses in Abhängigkeit von $P(x=+1)$ bzw. y

Im linken Bild fällt zunächst auf, dass der Verlauf symmetrisch zum Punkt (0,5;0) ist. Hier findet ein Vorzeichenwechsel statt. Für $P(x=+1)>0,5$ ist die Wahrscheinlichkeit für eine '+1' größer als für eine '-1', so dass $L_a(x)$ ab hier positive Werte annimmt. Je größer die Differenz zwischen $P(x=+1)$ und $P(x=-1)$ ist, desto größer werden die L-Werte, was belegt, dass sie ein geeignetes Maß für die Zuverlässigkeit eines Symbols sind. Sind '+1' und '-1' gleich wahrscheinlich ($P(x=+1)=0,5$), hat $L_a(x)$ den Wert Null, eine Entscheidung wäre rein zufällig. Trägt man $L_a(x)$ über $P(x=-1)$ auf, kehren sich lediglich die Vorzeichen um und man erhält einen zur Abszisse gespiegelten Verlauf.

Für den einfachen AWGN-Kanal und den 1-Pfad Rayleigh-Kanal nimmt der Term $L(y|x)$ aus Gl. (1.21) folgende konkrete Form an.

$$\begin{aligned} L(y|x) &= \ln\frac{\exp\left(-\frac{(y-\alpha\sqrt{E_s/T_s})^2}{2\sigma^2}\right)}{\exp\left(-\frac{(y+\alpha\sqrt{E_s/T_s})^2}{2\sigma^2}\right)}\,; \quad \sigma^2=\frac{N_0}{2T_s} \\ &= \frac{4\alpha\sqrt{E_s/T_s}y}{N_0/T_s} \\ &= \underbrace{4\alpha\frac{E_s}{N_0}}_{L_{ch}}y' \quad \text{mit} \quad y'=\frac{y}{\sqrt{E_s/T_s}} \end{aligned} \tag{1.22}$$

In Gl. (1.22) repräsentiert y' den auf $\sqrt{E_s/T_s}$ normierten Empfangswert y. Der Faktor α gibt den Fading-Koeffizienten an, der im Fall des AWGN-Kanals den Wert $\alpha=1$ besitzt. Der Koeffizient L_{ch} beschreibt die Zuverlässigkeit des Kanals, welche natürlich vom Signal-Rausch-Verhältnis E_s/N_0, aber auch von der Fading-Amplitude α abhängt. Für große L_{ch} ist der Kanal sehr zuverlässig, für kleine Werte besteht hingegen eine große Unsicherheit bzgl. des empfangenen Symbols. Diese Zusammenhänge illustriert der rechte Teil von Bild 1.19.

Aus Gl. (1.22) folgt, dass am Ausgang eines *matched*-Filters direkt die *Log-Likelihood*-Verhältnisse anliegen. Wenn wir in der Lage sind, eine geeignete Arithmetik für die LLR's zu finden, müssen wir diese nicht mehr

in Wahrscheinlichkeiten umrechnen, sondern können die empfangenen Werte direkt weiterverarbeiten. Eine solche Arithmetik wird von Hagenauer als L-**Algebra** bezeichnet [Hag96] und im nächsten Abschnitt noch eingehend behandelt.

Man kann natürlich aus den LLR's auch auf die Wahrscheinlichkeiten $P(x=+1)$ bzw. $P(x=-1)$ zurückrechnen. Es ergeben sich die folgenden Ausdrücke

$$P(x=+1|y) = \frac{e^{L(\hat{x})}}{1+e^{L(\hat{x})}} \tag{1.23}$$

$$P(x=-1|y) = \frac{1}{1+e^{L(\hat{x})}} \, . \tag{1.24}$$

Bezogen auf das Symbol x gilt

$$P(x=i|y) = \frac{e^{L(\hat{x})/2}}{1+e^{L(\hat{x})}} \cdot e^{iL(\hat{x})/2} \quad \text{mit} \quad i \in \{-1,+1\} \, . \tag{1.25}$$

Die Wahrscheinlichkeit für die Richtigkeit eines Empfangswertes $P(\hat{x}$ korrekt$)$ ist ebenfalls einfach zu bestimmen. Für $x=+1$ liegt eine korrekte Entscheidung vor, wenn $L(\hat{x})$ positiv ist, d.h.

$$P(\hat{x} \text{ korrekt}|x=+1) = \frac{e^{L(\hat{x})}}{1+e^{L(\hat{x})}} = \frac{e^{|L(\hat{x})|}}{1+e^{|L(\hat{x})|}} \, .$$

Für $x=-1$ muss $L(\hat{x})$ hingegen negativ sein und es folgt

$$P(\hat{x} \text{ korrekt}|x=-1) = \frac{1}{1+e^{L(\hat{x})}} = \frac{1}{1+e^{-|L(\hat{x})|}} = \frac{e^{|L(\hat{x})|}}{1+e^{|L(\hat{x})|}} \, .$$

Wir erhalten also in beiden Fällen das gleiche Ergebnis

$$P(x \text{ korrekt}) = \frac{e^{|L(\hat{x})|}}{1+e^{|L(\hat{x})|}} \, . \tag{1.26}$$

Ferner gilt für den Erwartungswert einer Datenentscheidung

$$\mathrm{E}\{\hat{x}\} = \sum_{i=\pm 1} i \cdot P(x=i) = \frac{e^{L(\hat{x})}}{1+e^{L(\hat{x})}} - \frac{1}{1+e^{L(\hat{x})}} = \tanh(L(\hat{x})/2) \, . \tag{1.27}$$

1.7.2 Rechnen mit *Log-Likelihood*-Werten (L-Algebra)

Wie schon aus dem letzten Semester bekannt ist, werden die Prüfbit eines Codes durch modulo-2-Addition bestimmter Informationsbit u_i berechnet. Damit gewinnt auch die Berechnung der L-Werte von verknüpften Zufallsvariablen an Bedeutung. Wir betrachten zunächst zwei **statistisch unabhängige** Symbole $x_1 = 1-2u_1$ und $x_2 = 1-2u_2$. Das LLR ihrer modulo-2-Summe berechnet sich nach

$$\begin{aligned}
L(u_1 \oplus u_2) &= \ln \frac{P(u_1 \oplus u_2 = 0)}{P(u_1 \oplus u_2 = 1)} \\
&= \ln \frac{P(x_1 \cdot x_2 = +1)}{P(x_1 \cdot x_2 = -1)} \\
&= \ln \frac{P(x_1=+1) \cdot P(x_2=+1) + P(x_1=-1) \cdot P(x_2=-1)}{P(x_1=+1) \cdot P(x_2=-1) + P(x_1=-1) \cdot P(x_2=+1)} \\
&= \ln \frac{P(x_1=+1)/P(x_1=-1) \cdot P(x_2=+1)/P(x_2=-1) + 1}{P(x_1=+1)/P(x_1=-1) + P(x_2=+1)/P(x_2=-1)} \\
L(x_1 \cdot x_2) &= \ln \frac{\exp\left(L(x_1)+L(x_2)\right)+1}{\exp\left(L(x_1)\right)+\exp\left(L(x_2)\right)} \, .
\end{aligned} \tag{1.28}$$

Mit Gl. (1.28) steht nun ein Funktional zur Verfügung, um das *log-likelihood*-Verhältnis der Verknüpfung zweier statistisch unabhängiger Größen zu berechnen. Die mathematischen Umrechnungen werden in der Literatur auch als L-**Algebra** bezeichnet. Gl. (1.28) lässt sich mit Hilfe der Beziehungen $\tanh(x/2) = (e^x - 1)/(e^x + 1)$ und $\ln \frac{1+x}{1-x} = 2\,\mathrm{arthanh}(x)$ in folgende Form umschreiben[1]

$$L(u_1 \oplus u_2) = \ln \frac{1 + \tanh(L(x_1)/2) \cdot \tanh(L(x_2)/2)}{1 - \tanh(L(x_1)/2) \cdot \tanh(L(x_2)/2)} \tag{1.29}$$

$$= 2\,\mathrm{arthanh}(\tanh(L(x_1)/2) \cdot \tanh(L(x_2)/2))$$

$$= 2\,\mathrm{arthanh}(\lambda_1 \cdot \lambda_2) \text{ mit } \lambda_i = \tanh(L(x_i)/2)\ . \tag{1.30}$$

Durch Gl. (1.30) ist eine einfache schaltungstechnische Realisierung möglich, wie sie in Bild 1.20 dargestellt ist. Die Eingangswerte $L(x_i)$ stellen die Ausgangssignale des *matched*-Filters dar, die dann über die tanh-Funktion nichtlinear abgebildet werden. Das Ergebnis der artanh-Funktion des Produktes stellt dann das gesuchte LLR dar.

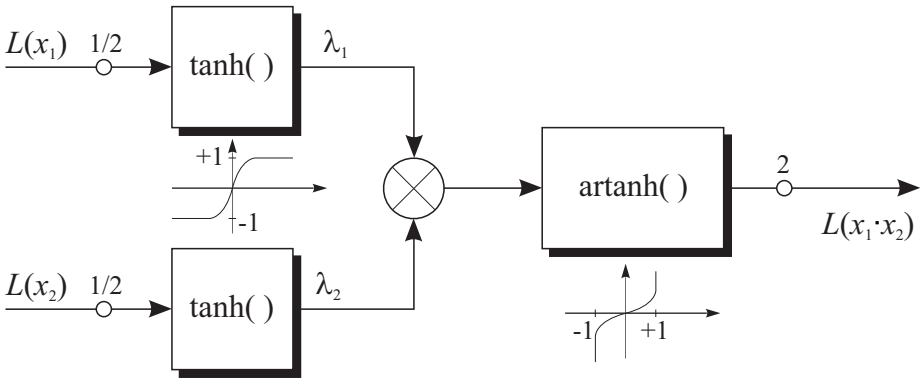

Bild 1.20: Berechnung des LLR für das Produkt zweier statistisch unabhängiger Signale

Betrachtet man den Verlauf der tanh-Funktion, so lässt sich einfach eine Approximation ableiten. Für betragsmäßig große LLR's gerät der tanh in die Sättigung, er strebt asymptotisch gegen ± 1. Dazwischen besitzt er einen annähernd linearen Verlauf mit einem Winkel von $\pi/4$ im Nullpunkt. Da das Produkt der λ_i gebildet wird, spielen Werte in der Nähe von ± 1 für den Betrag des Ergebnisses keine Rolle, dieser wird aufgrund der 'Fast-Linearität' durch das Minimum der Eingangsbeträge bestimmt. Das Vorzeichen ergibt sich hingegen aus dem Produkt der einzelnen Vorzeichen. Wir erhalten also folgende vereinfachende Approximation

$$L(u_1 \oplus u_2) \approx \mathrm{sgn}\{L(x_1)\} \cdot \mathrm{sgn}\{L(x_2)\} \cdot \min\{|L(x_1)|, |L(x_2)|\} \tag{1.31}$$

Allgemein können die Ausdrücke in den letzten Gleichungen auch für mehr als 2 Variablen angegeben werden. Ein Beweis für die Gültigkeit kann per vollständiger Induktion erfolgen (s. Übung Kanalcodierung 2). Im Folgenden sind für n statistisch unabhängige Symbole kurz die Ergebnisse aufgeführt.

$$L(u_1 \oplus \ldots \oplus u_n) = \ln \frac{\prod_{i=1}^{n}(e^{L(u_i)} + 1) + \prod_{i=1}^{n}(e^{L(u_i)} - 1)}{\prod_{i=1}^{n}(e^{L(u_i)} + 1) - \prod_{i=1}^{n}(e^{L(u_i)} - 1)} \tag{1.32}$$

$$= \ln \frac{1 + \prod_{i=1}^{n} \tanh(L(x_i)/2)}{1 - \prod_{i=1}^{n} \tanh(L(x_i)/2)} = 2\mathrm{arthanh}\left(\prod_{i=1}^{n} \tanh(L(x_i)/2)\right) \tag{1.33}$$

$$\approx \prod_{i=1}^{n} \mathrm{sgn}\{L(x_i)\} \cdot \min_i\{|L(x_i)|\} \tag{1.34}$$

[1] Die Umformung lässt sich leichter zeigen, wenn Gl. (1.28) ausgehend von Gl. (1.30) abgeleitet wird.

1.7.3 Allgemeiner Ansatz zur *Soft-Output*-Decodierung

Betrachten wir nun den Fall einer Kanalcodierung. Das Ziel besteht darin, das bekannte MAP-Kriterium zu erfüllen. Anhand einer empfangenen Sequenz $\mathbf{y}$ soll eine Aussage über die Informationsbit $\mathbf{u}$ getroffen werden. Genauer gesagt: wir wollen anhand von $\mathbf{y}$ für jedes Bit u_i eine Entscheidung und die dazugehörige Zuverlässigkeit bestimmen. Gemäß dem sogenannten *Symbol-by-Symbol*-MAP-Kriterium muss das *Log-Likelihood*-Verhältnis

$$L(\hat{u}_i) = \ln \frac{p(u_i = 0, \mathbf{y})}{p(u_i = 1, \mathbf{y})} \tag{1.35}$$

berechnet werden. Die Verbundwahrscheinlichkeiten in Gl. (1.35) sind nicht direkt zugänglich. Sie müssen vielmehr mit Hilfe einiger elementarer Umformungen abgeleitet werden. Dabei hilft die Beziehung $P(a) = \sum_i P(a, b_i)$, denn wir teilen nun den gesamten Coderaum in 2 Teilmengen $\Gamma_i^{(1)}$ und $\Gamma_i^{(0)}$ auf. $\Gamma_i^{(1)}$ enthält nur die Codeworte $\mathbf{c}$, deren i-tes Informationssymbol eine '1' ist ($u_i = 1$). Für $\Gamma_i^{(0)}$ gilt entsprechend $u_i = 0$. Wir erhalten

$$L(\hat{u}_i) = \ln \frac{\sum\limits_{\mathbf{c} \in \Gamma_i^{(0)}} p(\mathbf{c}, \mathbf{y})}{\sum\limits_{\mathbf{c} \in \Gamma_i^{(1)}} p(\mathbf{c}, \mathbf{y})} \tag{1.36}$$

$$= \ln \frac{\sum\limits_{\mathbf{c} \in \Gamma_i^{(0)}} p(\mathbf{y}|\mathbf{c}) \cdot P(\mathbf{c})}{\sum\limits_{\mathbf{c} \in \Gamma_i^{(1)}} p(\mathbf{y}|\mathbf{c}) \cdot P(\mathbf{c})} \tag{1.37}$$

Für den AWGN-Kanal sind alle aufeinander folgenden Rauschwerte statistisch unabhängig voneinander, was für die Binärstellen eines Codewortes natürlich nicht gilt (die Codierung fügt statistische Abhängigkeiten ein). Die bedingte Wahrscheinlichkeitsdichte $p(\mathbf{y}|\mathbf{c})$ stellt jedoch die Wahrscheinlichkeitsdichte unter der Hypothese $\mathbf{c}$ dar ($\mathbf{c}$ ist keine Zufallsgröße mehr, sondern eine feste Annahme). Deswegen können die Elemente y_i von $\mathbf{y}$ als statistisch unabhängig angesehen werden (die statistischen Abhängigkeiten sind in der Hypothese $\mathbf{c}$ enthalten). Die bedingte Wahrscheinlichkeitsdichte der Vektoren kann dann in das Produkt der Dichten der Vektorelemente überführt werden.

$$L(\hat{u}_i) = \ln \frac{\sum\limits_{\mathbf{c} \in \Gamma_i^{(0)}} \prod\limits_{j=0}^{n-1} p(y_j|c_j) \cdot P(\mathbf{c})}{\sum\limits_{\mathbf{c} \in \Gamma_i^{(1)}} \prod\limits_{j=0}^{n-1} p(y_j|c_j) \cdot P(\mathbf{c})} \tag{1.38}$$

Außerdem ist ein Codewort $\mathbf{c}$ eindeutig durch seine Informationsbit $\mathbf{u}$ bestimmt, wodurch für die Auftrittswahrscheinlichkeit $P(\mathbf{c}) = P(\mathbf{u})$ gilt. Die Informationsbit u_i sind allerdings statistisch unabhängig voneinander (**nicht die Codebit** c_i), so dass $P(\mathbf{c}) = \prod_{i=0}^{k-1} P(u_i)$ gilt.

$$L(\hat{u}_i) = \ln \frac{\sum\limits_{\mathbf{c} \in \Gamma_i^{(0)}} \prod\limits_{j=0}^{n-1} p(y_j|c_j) \cdot \prod\limits_{j=0}^{k-1} P(u_j)}{\sum\limits_{\mathbf{c} \in \Gamma_i^{(1)}} \prod\limits_{j=0}^{n-1} p(y_j|c_j) \cdot \prod\limits_{j=0}^{k-1} P(u_j)} \tag{1.39}$$

Für systematische Codierer gilt $u_i = c_i$, weshalb der mit der i-ten Stelle korrespondierende Term $p(y_i|c_i)$ in Zähler und Nenner jeweils konstant ist. Daher kann er zusammen mit $P(u_i)$ aus dem Produkt und der Summe

herausgezogen werden

$$L(\hat{u}_i) = \underbrace{\ln \frac{p(y_i|u_i=0)}{p(y_i|u_i=1)}}_{L_{ch}\cdot y_i} + \underbrace{\ln \frac{P(u_i=0)}{P(u_i=1)}}_{L_a(u_i)} + \underbrace{\ln \frac{\sum\limits_{\mathbf{c}\in\Gamma_i^{(0)}} \prod\limits_{\substack{j=0\\ j\neq i}}^{n-1} p(y_j|c_j) \cdot \prod\limits_{\substack{j=0\\ j\neq i}}^{k-1} P(u_j)}{\sum\limits_{\mathbf{c}\in\Gamma_i^{(1)}} \prod\limits_{\substack{j=0\\ j\neq i}}^{n-1} p(y_j|c_j) \cdot \prod\limits_{\substack{j=0\\ j\neq i}}^{k-1} P(u_j)}}_{L_e(\hat{u}_i)} . \tag{1.40}$$

Aus Gl. (1.40) geht hervor, dass sich $L(\hat{u}_i)$ bei einer systematischen Codierung aus drei Anteilen zusammensetzt:

$$\boxed{L(\hat{u}_i) = L_{ch} y_i + L_a(u_i) + L_e(\hat{u}_i) \ ,} \tag{1.41}$$

dem *log-likelihood*-Verhältnis des direkt empfangenen Symbols y_i - also der systematischen Komponente -, der a-priori-Information $L_a(u_i)$ (schon vom uncodierten Fall bekannt) und einem Anteil $L_e(\hat{u}_i)$, der nicht von u_i bzw. y_i selbst abhängt. Dieser wird vielmehr 'von außen' aus allen durch die Codierung mit u_i verknüpften Bit berechnet und daher **extrinsische Information** genannt. Erfahren die einzelnen Codesymbole während der Übertragung statistisch unabhängige Störungen (wie z.B. beim AWGN-Kanal), so ist $L_e(u_i)$ statistisch unabhängig von $L_a(u_i)$ und $L_{ch}y_i$ und liefert daher einen Beitrag zum Decodierergebnis, der die Zuverlässigkeit der Entscheidung erhöhen kann. **Diese Zerlegung kann nicht für nicht-systematische Codes vorgenommen werden!**

Wir können die extrinsische Information auch noch kompakter schreiben und erhalten

$$L_e(\hat{u}_i) = \ln \frac{\sum\limits_{\mathbf{c}\in\Gamma_i^{(0)}} \prod\limits_{\substack{j=0\\ j\neq i}}^{n-1} p(y_j;c_j)}{\sum\limits_{\mathbf{c}\in\Gamma_i^{(1)}} \prod\limits_{\substack{j=0\\ j\neq i}}^{n-1} p(y_j;c_j)} \quad \text{mit} \quad p(y_j;c_j) = \begin{cases} p(y_j|c_j)\cdot P(u_j) & \text{für } 0 \le j < k \\ p(y_j|c_j) & \text{für } k \le j < n \ . \end{cases} \tag{1.42}$$

Unter Zuhilfenahme der LLR's lautet die extrinsische Information

$$L_e(\hat{u}_i) = \ln \frac{\sum\limits_{\mathbf{c}\in\Gamma_i^{(0)}} \prod\limits_{\substack{j=0\\ j\neq i}}^{n-1} \exp\left[-L(c_j;y_j)\cdot c_j\right]}{\sum\limits_{\mathbf{c}\in\Gamma_i^{(1)}} \prod\limits_{\substack{j=0\\ j\neq i}}^{n-1} \exp\left[-L(c_j;y_j))\cdot c_j\right]} \quad \text{mit} \quad L(c_l;y_l) = \begin{cases} L_{ch}y_l + L_a(u_l) & \text{für} \quad 0 \le l < k \\ L_{ch}y_l & \text{für} \quad k \le l < n \ . \end{cases} \tag{1.43}$$

Mit den Gleichungen (1.41) und (1.42) kann also eine Decodierung durchgeführt werden, die neben der reinen Hard-Decision auch ein Maß für die Zuverlässigkeit der Decodierentscheidung liefert. Ein Problem besteht allerdings darin, dass zur Berechnung der extrinsischen Information über alle Codeworte $\mathbf{c}$ des Coderaums Γ summiert werden muss. Man kann sich leicht vorstellen, dass diese direkte Realisierung der *Soft-Output*-Decodierung sehr aufwendig ist, insbesondere dann, wenn Codes mit sehr großen Alphabeten zum Einsatz kommen. Für einen (7,4,3)-Hamming-Code mit seinen $2^4 = 16$ Codeworten dürfte eine Berechnung von Gl. (1.42) kein Problem sein, für einen (255,247,3)-Hamming-Code gibt es allerdings $2^{247} = 2,3 \cdot 10^{74}$ Codeworte, so dass Gl. (1.42) selbst auf Hochleistungsrechnern nicht mehr zu berechnen ist.

Decodierung über den *dualen Code*

Ist die Anzahl der Prüfbit relativ klein, gibt es die Möglichkeit, die Decodierung über den *dualen Code* durchzuführen. Dieser ist aus dem letzten Semester noch bekannt und stellt den zum Originalcode C orthogonalen Code $C^{\perp}$ dar. Es soll an dieser Stelle nicht auf Einzelheiten dieser Decodierungsmöglichkeit eingegangen, die Decodiervorschrift jedoch kurz erläutert werden. Der Vorteil besteht darin, dass jetzt über alle Codewörter aus

$C^{\perp}$ summiert werden muss. Für den oben erwähnten (255,247,3)-Hamming-Code wären das nur $2^8 = 256$, was zu einer deutlichen Vereinfachung führt. Das Decodierergebnis lautet [Off96]

$$L(\hat{u}_i) = L_{ch} y_i + L_a(u_i) + \ln \frac{\sum\limits_{\mathbf{c}' \in \Gamma^{\perp}} \prod\limits_{\substack{l=0 \\ l \neq i}}^{n-1} [\tanh(L(y_l; c_l)/2)]^{c'_l}}{\sum\limits_{\mathbf{c}' \in \Gamma^{\perp}} (-1)^{c'_i} \prod\limits_{\substack{l=0 \\ l \neq i}}^{n-1} [\tanh(L(y_l; c_l)/2)]^{c'_l}} . \tag{1.44}$$

In Gl. (1.44) stellt c'_l stellt das l-te Codebit des Codewortes $\mathbf{c}'$ aus $C^{\perp}$ dar. Es wird also in Zähler und Nenner über alle 2^{n-k} Codeworte des dualen Codes $C^{\perp}$ summiert, wobei die einzelnen Summanden sich aus dem Produkt der tanh-Terme der einzelnen Codebit zusammensetzen. Eine weitere Möglichkeit zur Reduktion des Rechenaufwandes besteht in der Ausnutzung der Markov-Eigenschaft von Codes, die in der Darstellung eines Trellisdiagramms resultiert.

Der Ansatz über den dualen Code führt allerdings auch nur dann zum Ziel, wenn die Anzahl der Redundanzbit gering ist. Sonst ergibt sich das gleiche Problem eines nicht zu bewältigenden Rechenaufwandes wie schon bei der direkten Decodierung des Originalcodes. Für diese Fälle gibt es eine Reihe von suboptimalen Algorithmen, die allerdings noch Gegenstand der aktuellen Forschung sind. Unter anderem wird versucht, leistungsfähige Codes zu finden, die einen geringen Decodieraufwand erfordern (*Low Density Parity Check Codes*). Im Rahmen dieser Vorlesung soll auf diese Verfahren nicht weiter eingegangen werden.

***Soft-Output*-Decodierung am Beispiel eines (4,3,2)-SPC-Codes**
Wir wollen nun am Beispiel eines einfachen (4,3,2)-SPC-Codes die L-Algebra genauer betrachten (s. Bild 1.21). Gegeben ist ein Informationswort

$$\mathbf{u} = (1\ 0\ 1) ,$$

das zunächst codiert ($\mathbf{c}$) und BPSK-moduliert wird

$$\mathbf{c} = (1\ 0\ 1\ 0) \quad \longrightarrow \quad \mathbf{x} = (-1\ +1\ -1\ +1) .$$

Während der Übertragung z.B. über einen AWGN-Kanal erfährt $\mathbf{x}$ Störungen, so dass der Vektor

$$\mathbf{y} = (-0,8\ +1,1\ \underline{+0,3}\ +0,4)$$

empfangen wird. Das Vorzeichen der Stelle x_2 wurde also verfälscht, zusätzlich sind die Beträge verrauscht. Wir gehen zunächst davon aus, dass keine a-priori-Information zur Verfügung steht, d.h. $L_a(u_i) = 0,\ i = 0, 1, 2$.

Bei einer einfachen Hard-Decision-Decodierung können wir den obigen Einzelfehler lediglich erkennen, allerdings nicht korrigieren.

Für eine Soft-Decision-Decodierung berechnen wir zunächst die LLR's für jedes y_i entsprechend Gl. (1.22) und erhalten für einen angenommenen Signal-Rausch-Abstand von 2 dB ($L_{ch} = 6,34$)

$$L_{ch}\mathbf{y} = (-5,1\ +7,0\ +1,9\ +2,5) .$$

Anschließend werden die statistischen Bindungen der einzelnen Symbole, die durch die Codierung eingebracht wurden, ausgenutzt und die extrinsische Information aus Gl. (1.41) bestimmt. Dazu stellen wir folgende Betrachtung an.

Bei einer Decodierung über den originalen Code nach Gl. (1.43) wären für den einfachen Parity-Check-Code $2^3 = 8$ Codeworte zu betrachten. Demgegenüber besteht der orthogonale Coderaum $C^{\perp}$ nur aus $2^{n-k} = 2$ Elementen, die Berechnung von $L_e(\hat{u}_i)$ über den dualen Code erfordert also nur die Berücksichtigung einer Prüfgleichung (das Nullwort wird nicht benötigt).

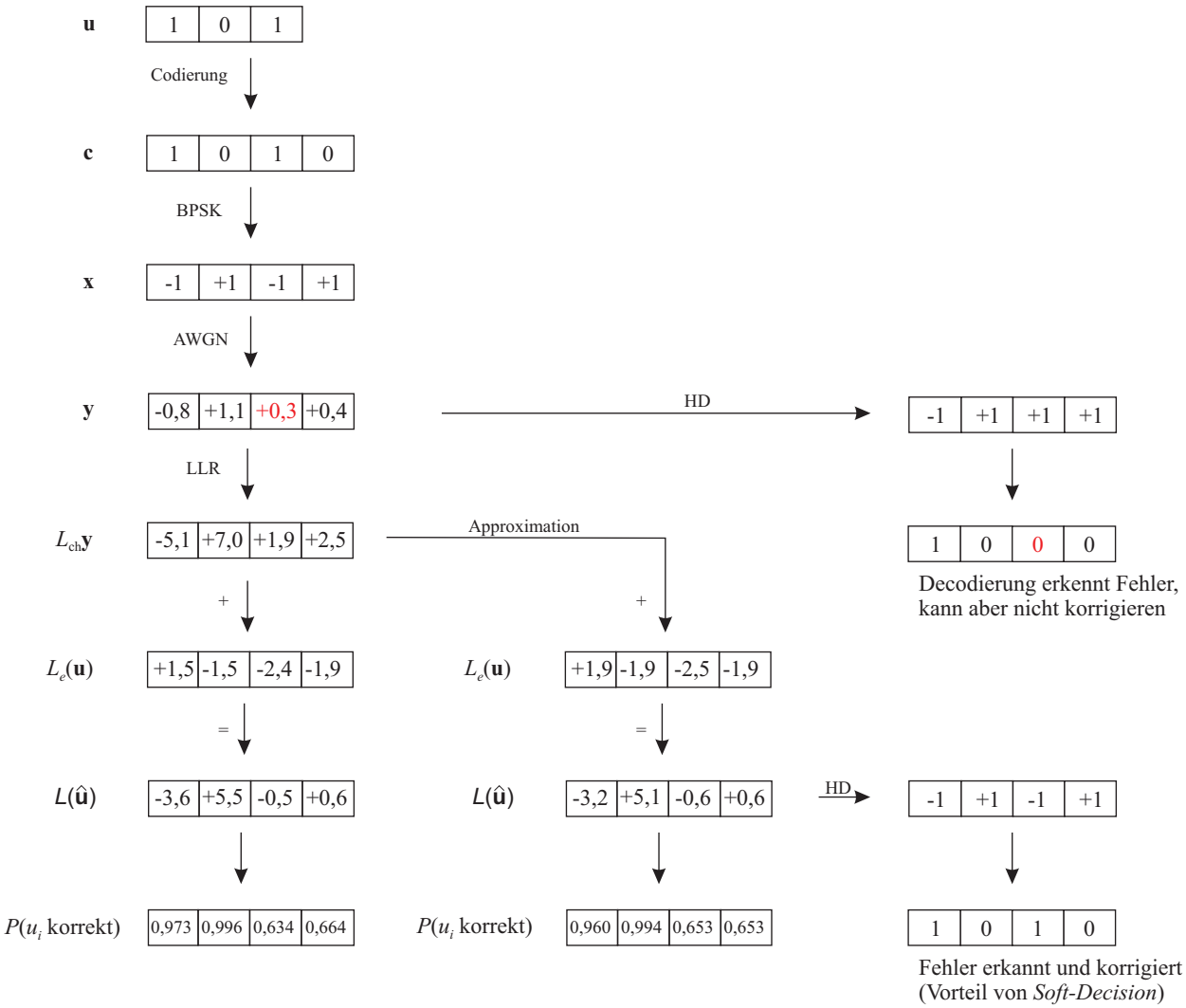

Bild 1.21: Darstellung der Soft-In/Soft-Out-Decodierung für einfachen SPC-Code

Da Zähler und Nenner in Gl. (1.44) für das Nullwort Eins ergeben, ist für SPC-Codes nur noch das Codewort $\mathbf{c}' = (1\,1\,\ldots\,1) \in \Gamma^{\perp}$ zu betrachten. Gl. (1.44) reduziert sich damit auf

$$L(\hat{u}_i) = L_{ch} y_i + \ln \frac{1 + \prod\limits_{\substack{l=0 \\ l \neq i}}^{n-1} [\tanh(L(y_l; c_l)/2)]}{1 - \prod\limits_{\substack{l=0 \\ l \neq i}}^{n-1} [\tanh(L(y_l; c_l)/2)]} \; . \tag{1.45}$$

Mit Hilfe der Beziehung $\ln \frac{1+x}{1-x} = 2\,\mathrm{arthanh}(x)$ erhalten wir schließlich (vgl. auch Gl. (1.33))

$$L(\hat{u}_i) = L_{ch} y_i + 2\,\mathrm{arthanh}\left(\prod_{\substack{l=0 \\ l \neq i}}^{n-1} [\tanh(L(y_l; c_l)/2)] \right) \tag{1.46}$$

$$\approx L_{ch} y_i + \prod_{\substack{l=0 \\ l \neq i}}^{n-1} \mathrm{sgn}\{L(y_l; c_l)\} \cdot \min_{l \neq i} \{|L(y_l; c_l)|\} \; . \tag{1.47}$$

Gl. (1.46) ist folgendermaßen zu interpretieren. Für alle $\mathbf{c} \in \Gamma$ gilt bekanntlich $\mathbf{c} \cdot \mathbf{c}' = 0$, d.h. die Quersumme der Binärstellen von $\mathbf{c}$ modulo 2 ergibt immer Null. Damit lässt sich aber jedes Bit c_i aus der modulo-2-Summe

aller übrigen Binärstellen berechnen $c_i = (\sum_{j \neq i} c_j) \mod 2$. Soll diese Berechnung nun mit Soft-Werten durchgeführt werden, ist Gl. (1.33) anzuwenden, wir erhalten mit dem zweiten Term in Gl. (1.46) also die extrinsische Information über das Bit u_i.

Für u_0 lautet die extrinsische Information

$$L_e(\hat{u}_0) = L_e(\hat{c}_0) = L(c_1 \oplus c_2 \oplus c_3)$$

und kann nach den Gleichungen (1.32), (1.33) oder (1.34) berechnet werden. Die gleichen Berechnungen werden für die übrigen Informationsbit u_2 und u_3 ausgeführt, so dass wir mit der Näherung aus Gl. (1.34)

$$L_e(\hat{\mathbf{u}}) = (+1,9 \ -1,9 \ -2,5 \ -1,9)$$

erhalten.

Es ist zu erkennen, dass durch die Verfälschung des Vorzeichens von x_2 für alle Stellen bis auf $i = 2$ die extrinsische Information eine anderes Ergebnis (Vorzeichen) liefert als die direkt empfangenen Werte. Damit kann sie das Decodierergebnis auch für diese Symbole verfälschen, vorausgesetzt, ihr Betrag ist größer als der der direkt empfangenen Werte. Auf der anderen Seite besteht die Möglichkeit, das verfälsche Vorzeichen von c_2 mit Hilfe von $L_e(\hat{u}_2)$ zu korrigieren.

Da $L_e(\hat{u}_i)$ statistisch unabhängig von den direkten Komponenten $L_{ch}y_i$ ist, können beide LLR's addiert werden. Ob die extrinsische Information dann das Vorzeichen des in unserem Beispiel verfälschten Bits c_2 korrigieren kann und die übrigen (korrekten) nicht mehr verändert, hängt von den Beträgen der Summanden ab. Das Ergebnis lautet

$$L(\hat{\mathbf{u}}) = L_{ch} \cdot \mathbf{y} + L_e(\hat{\mathbf{u}}) = (-3,2 \ +5,1 \ -0,6 \ +0,6) \quad \longrightarrow \quad \mathbf{c} = (1\,0\,1\,0) \ .$$

Wir erkennen, dass die Vorzeichen zwar korrekt sind, der Betrag für $\hat{u}_2$ ist allerdings deutlich kleiner als bei den übrigen Stellen und die Entscheidung somit unsicherer. Dies wird klarer, wenn man sich die zu den LLR's zugehörigen Wahrscheinlichkeiten anschaut (s. Bild 1.21). Die Wahrscheinlichkeit für die Richtigkeit der Vorzeichen beträgt dort nämlich nur $63,5\%$, während sie für die restlichen Stellen nahe Eins liegt.

Das bessere Ergebnis gegenüber der Hard-Decision-Decodierung besteht nun nicht in der Verwendung von *Soft-Output*-Algorithmen, sondern in der Ausnutzung der weichen Ausgangswerte des Kanals (*Soft-Decision*-Decodierung). Die weichen Ausgangswerte des Decodierers kommen erst zu Geltung, wenn weitere nachfolgende Stufen der Signalverarbeitung gewinnbringend Gebrauch von ihnen machen können. Dies wird der nächste Abschnitt verdeutlichen.

1.7.4 BCJR-Algorithmus am Beispiel von Faltungscodes

Der BCJR-Algorithmus wurde erstmals im Jahr 1972 von Bahl, Cocke, Jelinek und Raviv vorgestellt. Er stellt eine allgemeine Vorschrift zur *Symbol-by-Symbol*-MAP-Decodierung dar, die nicht nur zur Decodierung, sondern auch zur Entzerrung gedächtnisbehafteter Kanäle eingesetzt werden kann. Der Vorteil des BCJR-Algorithmus gegenüber der direkten Realisierung von Gl. (1.43) besteht in der effizienten Ausnutzung der Markov-Eigenschaft des Kanals bzw. des Codierers. Er setzt wie schon der Viterbi-Algorithmus die Darstellung des Systems (in unserem Fall des Kanalcodes) durch ein Trellisdiagramm voraus. Es gibt zwar auch eine Trellisrepräsentation für Blockcodes, diese müßte hier jedoch erst neu eingeführt werden. Aus diesem Grund greifen wir auf binäre $1/n$-ratige Faltungscodes zurück, für die das Trellisdiagramm schon bekannt ist (s. Kanalcodierung I).

Ausgangspunkt zur *Symbol-by-Symbol*-MAP-Decodierung ist wiederum Gl. (1.35)

$$L(\hat{u}_i) = \ln \frac{p(u_i = 0, \mathbf{y})}{p(u_i = 1, \mathbf{y})} \ .$$

Wir betrachten nun den Ausschnitt eines Trellisdiagramms für einen rekursiven, systematischen Faltungscodes der Einflusslänge $L_c = 3$. Aus Gründen der Anschaulichkeit wird in den Bildern stets dieser einfache RSC-Code mit nur 4 Zuständen verwendet, die mathematische Herleitung jedoch allgemein gehalten.

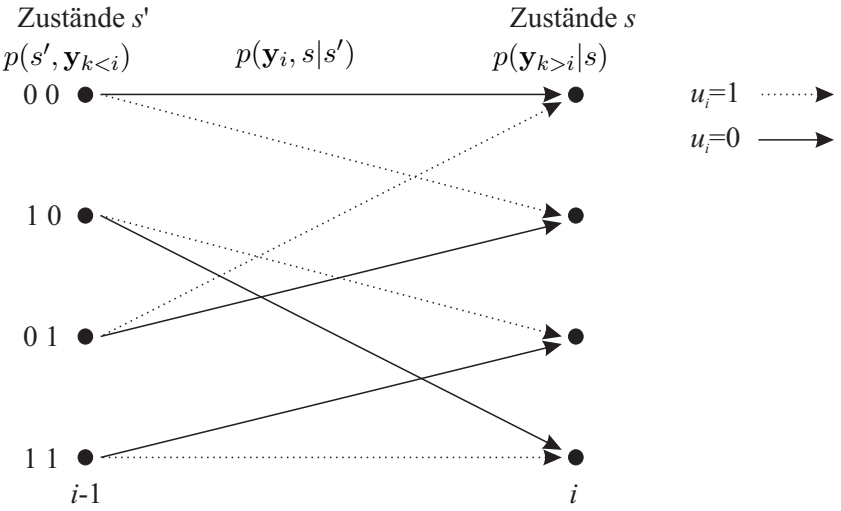

Bild 1.22: Ausschnitt eines Trellisdiagramms zur Erläuterung des BCJR-Algorithmus

Vom Zeitpunkt $i-1$ zum Zeitpunkt i findet durch das Informationsbit u_i ein Übergang von Zustand s' zu Zustand s statt. Alle möglichen Zustandsübergänge $s' \rightarrow s$ lassen sich in zwei Klassen aufteilen, in solche, die mit $u_i = 0$ korrespondieren und die übrigen, die mit $u_i = 1$ verknüpft sind. Bezeichnen wir ein Zustandspaar mit (s', s), so können wir obige Gleichung in

$$L(\hat{u}_i) = \ln \frac{\sum\limits_{(s',s),u_i=0} p(s', s, \mathbf{y})}{\sum\limits_{(s',s),u_i=1} p(s', s, \mathbf{y})} \tag{1.48}$$

umschreiben. Nun wird der Empfangsvektor $\mathbf{y}$ in drei Anteile zerlegt, einen Anteil $\mathbf{y}_{k<i}$, der alle empfangenen Symbole vor dem betrachteten Zeitpunkt i enthält, den aktuellen Empfangsvektor $\mathbf{y}_i = (y_{i,0}, \cdots, y_{i,n-1})$ bestehend aus n Empfangswerten und einen Anteil $\mathbf{y}_{k>i}$, der alle nach $\mathbf{y}_i$ empfangenen Symbole beinhaltet.

$$\begin{aligned} L(\hat{u}_i) &= \ln \frac{\sum\limits_{(s',s),u_i=0} p(s', s, \mathbf{y}_{k<i}, \mathbf{y}_i, \mathbf{y}_{k>i})}{\sum\limits_{(s',s),u_i=1} p(s', s, \mathbf{y}_{k<i}, \mathbf{y}_i, \mathbf{y}_{k>i})} \\ &= \ln \frac{\sum\limits_{(s',s),u_i=0} p(\mathbf{y}_{k>i}|s', s, \mathbf{y}_{k<i}, \mathbf{y}_i) \cdot p(s', s, \mathbf{y}_{k<i}, \mathbf{y}_i)}{\sum\limits_{(s',s),u_i=1} p(\mathbf{y}_{k>i}|s', s, \mathbf{y}_{k<i}, \mathbf{y}_i) \cdot p(s', s, \mathbf{y}_{k<i}, \mathbf{y}_i)} \end{aligned} \tag{1.49}$$

Für den ersten Faktor in Zähler und Nenner gilt: Ist der Zustand s zum Zeitpunkt i bekannt, so sind alle übrigen Größen (s', $\mathbf{y}_i$, $\mathbf{y}_{k<i}$) für den Vektor $\mathbf{y}_{k>i}$ ohne Belang, so dass

$$\beta_i(s) := p(\mathbf{y}_{k>i}|s', s, \mathbf{y}_{k<i}, \mathbf{y}_i) = p(\mathbf{y}_{k>i}|s) \tag{1.50}$$

gilt. $\beta_i(s)$ gibt die Wahrscheinlichkeitsdichte für die Teilfolge $\mathbf{y}_{k>i}$ an, wenn zum Zeitpunkt i der Zustand s im

Trellisdiagramm angenommen wird. Damit lautet Gl. (1.49) jetzt

$$\begin{aligned} L(\hat{u}_i) &= \ln \frac{\sum\limits_{(s',s),u_i=0} \beta_i(s) \cdot p(s, \mathbf{y}_i | s', \mathbf{y}_{k<i}) \cdot p(s', \mathbf{y}_{k<i})}{\sum\limits_{(s',s),u_i=1} \beta_i(s) \cdot p(s, \mathbf{y}_i | s', \mathbf{y}_{k<i}) \cdot p(s', \mathbf{y}_{k<i})} \\ &= \ln \frac{\sum\limits_{(s',s),u_i=0} \beta_i(s) \cdot p(s, \mathbf{y}_i | s') \cdot p(s', \mathbf{y}_{k<i})}{\sum\limits_{(s',s),u_i=1} \beta_i(s) \cdot p(s, \mathbf{y}_i | s') \cdot p(s', \mathbf{y}_{k<i})} \end{aligned} \tag{1.51}$$

Der mittlere Faktor in Zähler und Nenner von Gl. (1.51) wurde in der gleichen Art und Weise abgeleitet wie schon β. Er beinhaltet die Übergangswahrscheinlichkeit des Kanals, nämlich die Wahrscheinlichkeit des Auftretens von y_i bei bekanntem Zustandswechsel von s' nach s. Es gilt

$$\begin{aligned} \gamma_i(s', s) &:= p(s, \mathbf{y}_i | s') = p(s', s, \mathbf{y}_i)/P(s') = p(\mathbf{y}_i | s', s) \cdot \frac{P(s', s)}{P(s')} \\ &= p(\mathbf{y}_i | s', s) \cdot P(s | s') \; . \end{aligned} \tag{1.52}$$

Der Faktor $p(\mathbf{y}_i|s', s)$ beschreibt die Übergangswahrscheinlichkeit des Kanals, während $P(s|s')$ ein a-priori-Wissen verkörpert. Da $u_i = 0$ und $u_i = 1$ in der Regel gleichwahrscheinlich sind, treten auch die Übergänge $s' \rightarrow s$ (sie korrespondieren mit $u = 0$ bzw. $u = 1$) mit der gleichen Häufigkeit auf, es gilt $P(s|s') = 1/2$. Besitzt der Decodierer allerdings a-priori-Wissen über ein Informationsbit u_i, so ist dies gleichbedeutend mit einem Wissen über die Wahrscheinlichkeit der Übergänge $s' \rightarrow s$ zum Zeitpunkt i. Dieses Wissen kann über die Größe γ entsprechend Gl. (1.52) in den Decodierprozeß eingebracht werden.

Abschließend bleibt noch die Verbundwahrscheinlichkeitsdichte $p(s', \mathbf{y}_{k<i})$ zu erwähnen, die mit

$$\alpha_{i-1}(s') := p(s', \mathbf{y}_{k<i}) \tag{1.53}$$

abgekürzt wird. Es ergibt sich also der kompakte Ausdruck

$$L(\hat{u}_i) = \ln \frac{\sum\limits_{(s',s),u_i=0} \alpha_{i-1}(s') \cdot \gamma_i(s', s) \cdot \beta_i(s)}{\sum\limits_{(s',s),u_i=1} \alpha_{i-1}(s') \cdot \gamma_i(s', s) \cdot \beta_i(s)} \; . \tag{1.54}$$

Die Summanden aus Gl. (1.48) lassen sich also in drei Anteile gliedern, wobei $\alpha_{i-1}(s')$ die Zeitpunkte $k < i$, $\gamma_i(s', s)$ den aktuellen Zeitpunkt und $\beta_i(s)$ alle nachfolgenden Zeitpunkte $k > i$ abdeckt. α und β lassen sich rekursiv berechnen, wie folgende Rechnung zeigt.

$$\begin{aligned} \alpha_i(s) &= p(s, \mathbf{y}_{k<i+1}) = \sum_{s'} p(s', s, \mathbf{y}_{k<i+1}) = \sum_{s'} p(s', s, \mathbf{y}_{k<i}, \mathbf{y}_i) \\ &= \sum_{s'} p(s, \mathbf{y}_i | s', \mathbf{y}_{k<i}) \cdot p(s', \mathbf{y}_{k<i}) = \sum_{s'} p(s, \mathbf{y}_i | s') \cdot p(s', \mathbf{y}_{k<i}) \\ &= \sum_{s'} \gamma_i(s', s) \cdot \alpha_{i-1}(s') \end{aligned} \tag{1.55}$$

Gl. (1.55) illustriert, dass sich die α durch eine Vorwärtsrekursion sukzessive berechnen lassen, da $\alpha_i(s)$ zum aktuellen Zeitpunkt i von den vorangegangenen $\alpha_{i-1}(s')$ des Zeitpunktes $i-1$ abhängt. Für β gilt entsprechend

$$\begin{aligned}\beta_{i-1}(s') &= p(\mathbf{y}_{k>i-1}|s') = \sum_s p(s', s, \mathbf{y}_i, \mathbf{y}_{k>i})/P(s') \\ &= \sum_s p(\mathbf{y}_{k>i}|s', s, \mathbf{y}_i) \cdot \frac{p(s', s, \mathbf{y}_i)}{P(s')} = \sum_s p(\mathbf{y}_{k>i}|s) \cdot p(\mathbf{y}_i, s|s') \\ &= \sum_s \gamma_i(s', s) \cdot \beta_i(s) \,, \end{aligned} \tag{1.56}$$

d.h. β wird mit einer Rückwärtsrekursion bestimmt (s. Bild 1.23).

Initialisierung
Zur Realisierung des *Symbol-by-Symbol*-MAP-Algorithmus ist das Trellisdiagramm demnach zweimal abzuarbeiten. Zunächst findet eine Vorwärtsrekursion statt, die zeitgleich mit dem Empfang der Kanalsymbole ablaufen kann. Da wir als Startzustand zum Zeitpunkt $i = 0$ stets den Nullzustand annehmen, sind die Werte α wie folgt zu initialisieren:

$$\alpha_0(s') = \begin{cases} 1 & s' = 0 \\ 0 & s' \neq 0 \,. \end{cases} \tag{1.57}$$

Anschließend ist die Rückwärtsrekursion zur Bestimmung der β durchzuführen. Hier müssen 2 Fälle unterschieden werden. Ist der Endzustand zum Zeitpunkt N dem Decodierer bekannt (z.B. der Nullzustand durch Anfügen von *Tailbit*), lautet die Initialisierung

$$\beta_N(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \,. \end{cases} \tag{1.58}$$

Ist der Endzustand unbekannt, kann nach

$$\beta_N(s) = \alpha_N(s) \tag{1.59}$$

oder

$$\beta_N(s) = 2^{-m} \tag{1.60}$$

initialisiert werden, wobei m das Gedächtnis des Faltungscodes repräsentiert. Die prinzipielle Abarbeitung des Trellisdiagramms illustriert noch einmal Bild 1.23.

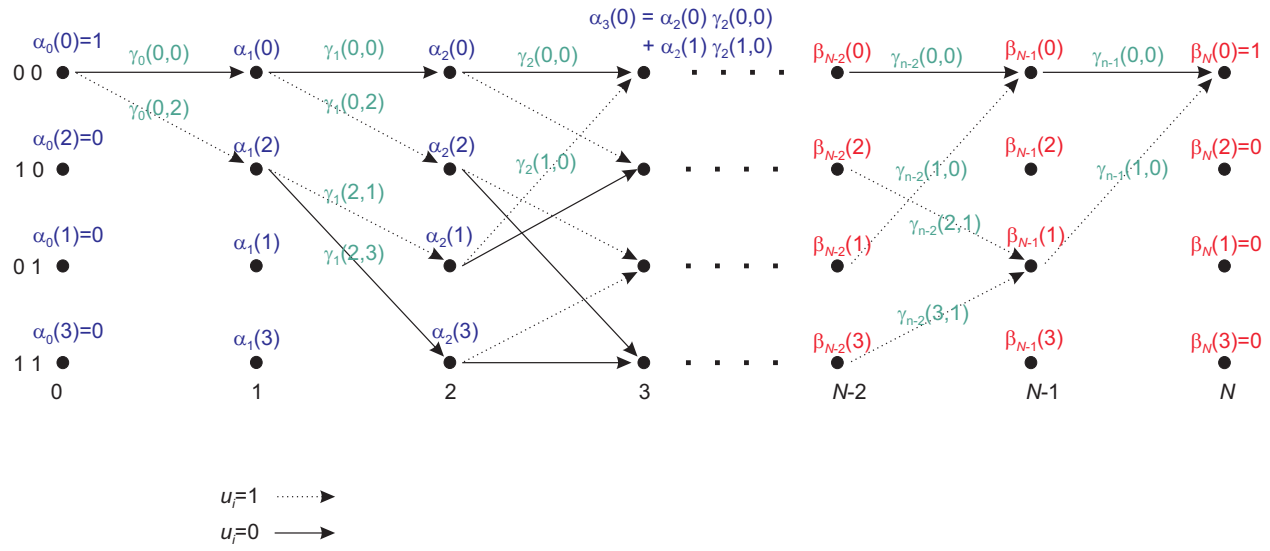

Bild 1.23: Abarbeitung des Trellisdiagramms durch BCJR-Algorithmus

1.7.5 Iterative ('Turbo')-Decodierung am Beispiel zweier parallel verketteter (5,4,2)-SPC-Codes

Die in den vorangegangenen Abschnitten vorgestellten *Soft-Output*-Algorithmen bieten nur dann Vorteile, wenn nachgeschaltete Module von der Zuverlässigkeitsinformation Gebrauch machen können. In den hier behandelten verketteten Codiersystemen erfolgt die Decodierung in der Regel durch serielle Aneinanderreihung der Teildecodierer, und zwar unabhängig davon, ob eine serielle oder eine parallele Codeverkettung vorliegt. Die prinzipielle Vorgehensweise soll zunächst anhand eines konkreten Produktcodes erläutert und dann in einer allgemeineren Form präsentiert werden.

Gegeben sei die folgende (4x4)-Matrix $\mathbf{u}$ bestehend aus 16 Informationsbit

$$\mathbf{u} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .$$

Sie wird sowohl horizontal als auch vertikal mit einem (5,4,2)-SPC-Code codiert, was zusammen mit der BPSK-Modulation zur Codematrix

$$\mathbf{x} = \begin{pmatrix} -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & \end{pmatrix}$$

führt. Es ist zu beachten, dass die rechte untere Ecke der Matrix unbesetzt bleibt, es handelt sich also um eine parallele Verkettung der beiden SPC-Codes (unvollständiger Produktcode). Durch die auf dem Kanal einwirkenden Störungen erhalten wir die Empfangsmatrix $\mathbf{y}$, deren Elemente entsprechend Gl. (1.22) in *log-likelihood*-Verhältnisse umgerechnet werden. ($E_b/N_0 = 2dB$).

$$\mathbf{y} = \left(\begin{array}{rrrr|r} 0,1 & 1,2 & 0,2 & -0,5 & 1,0 \\ 0,8 & -0,7 & 0,6 & -0,1 & -1,5 \\ -1,2 & 0,5 & -0,9 & 1,2 & 0,2 \\ 0,2 & -0,2 & 1,3 & -1,5 & -2,0 \\ \hline 0,3 & -0,9 & 1,2 & -1,1 & \end{array}\right) \Longrightarrow L_{ch}\mathbf{y} = \left(\begin{array}{rrrr|r} 0,6 & 7,6 & 1,3 & -3,2 & 6,3 \\ 5,1 & -4,4 & 3,8 & -0,6 & -9,5 \\ -7,6 & 3,2 & -5,7 & 7,6 & 1,3 \\ 1,3 & -1,3 & 8,2 & -9,5 & -12,7 \\ \hline 1,9 & -5,7 & 7,6 & -7,0 & \end{array}\right)$$

Diese Matrix bildet nun den Ausgangspunkt für einen **iterativen Decodierprozeß**, der im Folgenden beschrieben wird und mit der vertikalen Decodierung beginnt. Wir wissen aus Gl. (1.41)

$$L(\hat{u}_i) = L_{ch}y_i + L_a(u_i) + L_e(\hat{u}_i) ,$$

dass sich das LLR bei systematischen Codes aus drei Anteilen zusammensetzt, einer a-priori-Information $L_a(u_i)$, einem systematischen direkten Anteil $L_{ch}y_i$ und der extrinsischen Information $L_e(\hat{u}_i)$. Während $L_a(u_i)$ uns jetzt noch nicht zur Verfügung steht, stellt die Matrix $L_{ch}\mathbf{y}$ den direkten, schon berechneten Anteil dar und $L_e(\hat{u}_i)$ soll mit Hilfe von Gl. (1.34) berechnet werden. Wir erhalten nach der vertikalen Decodierung $D^{|}$ zu jedem Informationsbit eine extrinsische Information $L^{|}_{e,1}(\hat{u}_i)$, z.B.

$$\begin{aligned} L^{|}_{e,1}(\hat{u}_0) &= = \min(5,1;\ 7,6;\ 1,3;\ 1,9) \cdot (+1)(-1)(+1)(+1) = -1,3 \\ L^{|}_{e,1}(\hat{u}_1) &= = \min(0,6;\ 7,6;\ 1,3;\ 1,9) \cdot (+1)(-1)(+1)(+1) = -0,6 \\ &\vdots \end{aligned}$$

die zur direkten Komponente hinzu addiert wird und dann $L^{|}_1(\hat{u}_i)$ ergibt ($\mathbf{y}_s$ stellt den Anteil der Informationsbit von $\mathbf{y}$ dar).

1. Iteration, vertikale Decodierung

$L_{ch}\mathbf{y}$

0,6	7,6	1,3	-3,2	6,3
5,1	-4,4	3,8	-0,6	-9,5
-7,6	3,2	-5,7	7,6	1,3
1,3	-1,3	8,2	-9,5	-12,7
1,9	-5,7	7,6	-7,0	

$\Downarrow$

$L_{e,1}^{|}(\hat{\mathbf{u}})$

-1,3	-1,3	-3,8	-0,6	
-0,6	1,3	-1,3	-3,2	
0,6	-1,3	1,3	0,6	
-0,6	3,2	-1,3	-0,6	

$\Longrightarrow$

$L_1^{|}(\hat{\mathbf{u}}) = L_{ch}\mathbf{y}_s + L_{e,1}^{|}(\hat{\mathbf{u}})$

-0,7	6,3	-2,5	-3,8	
4,5	-3,1	2,5	-3,8	
-7,0	1,9	-4,4	8,2	
0,7	1,9	6,9	-10,1	

Damit ist die erste vertikale Decodierung abgeschlossen. Die berechnete extrinsische Information $L_{e,1}^{|}(\hat{u}_i)$ stellt das Wissen über ein bestimmtes Informationsbit u_i eines Codewortes aus der Sicht aller übrigen Bit $u_{j\neq i}$ dieses Codewortes dar, nicht aber aus der Sicht von u_i selbst. Da bei der horizontalen Decodierung D^- nun andere Binärstellen zu einem Codewort zusammengefaßt werden als bei der vertikalen, ist die extrinsische Information statistisch unabhängig von den Symbolen c_i eines horizontalen Codewortes und ein a-priori-Wissen für D^-.

$$L_{a,1}^{-}(\hat{\mathbf{u}}) = L_{e,1}^{|}(\hat{\mathbf{u}}) \tag{1.61}$$

Für die horizontale Decodierung wird daher $L_{a,1}^{-}(\hat{\mathbf{u}})$ zu jedem empfangenen LLR $L_{ch}\mathbf{y}$ hinzu addiert (beide Informationen beschreiben das gleiche Bit und sind statistisch unabhängig voneinander)

$$\begin{aligned} L_{ch}y_i + L_{e,1}^{|}(\hat{u}_i) \qquad &, \quad i = 0 \ldots 15 \\ L_{ch}y_0 + L_{e,1}^{|}(\hat{u}_0) &= 0,6 + (-1,3) = -0,7 \\ L_{ch}y_1 + L_{e,1}^{|}(\hat{u}_1) &= 5,1 + (-0,6) = 4,5 \end{aligned}$$

und wir erhalten folgende Eingangsmatrix für die erste horizontale Decodierung. Mit diesen 'neuen' Eingangswerten startet jetzt die horizontale Decodierung D^-.

1. Iteration, horizontale Decodierung

$L_{ch}\mathbf{y} + L_{a,1}^{-}(\mathbf{u})$

-0,7	6,3	-2,5	-3,8	6,3
4,5	-3,1	2,5	-3,8	-9,5
-7,0	1,9	-4,4	8,2	1,3
0,7	1,9	6,9	-10,1	-12,7
1,9	-5,7	7,6	-7,0	

$\Longrightarrow$

$\mathbf{L}_{e,1}^{-}(\hat{\mathbf{u}})$

2,5	-0,7	0,7	0,7	
-2,5	2,5	-3,1	2,5	
-1,3	1,3	-1,3	1,3	
1,9	0,7	0,7	-0,7	

$\Downarrow$

$\mathbf{L}_1^{-}(\hat{\mathbf{u}}) = L_{ch}\mathbf{y}_s + L_{a,1}^{-}(\mathbf{u}) + L_{e,1}^{-}(\hat{\mathbf{u}})$

1,8	5,6	-1,8	-3,1	
2,0	-0,6	-0,6	-1,3	
-8,3	3,2	-5,7	9,5	
2,6	2,6	7,6	-10,8	

1. Iteration, geschätzte Daten

Nach der ersten Iteration erhalten wir folgende geschätzten Daten

$$\hat{\mathbf{u}}_1 = \begin{array}{|c|c|c|c|}\hline 0&0&1&1\\\hline 0&1&1&1\\\hline 1&0&1&0\\\hline 0&0&0&1\\\hline\end{array}$$

Ein Vergleich mit der gesendeten Informationsmatrix $\mathbf{u}$ zeigt, dass zwei Fehler aufgetreten sind, in der ersten Zeile das erste und das dritte Bit. Allerdings ist das Decodierergebnis noch verbesserungsfähig, da der $D^{|}$ bisher noch nicht die extrinsische Information $L_{e,1}^{-}(\hat{\mathbf{u}})$ des horizontalen Decodierers als a-priori-Information ausnutzen konnte. Folglich führen wir nun eine zweite Iteration durch, in der $D^{|}$ wieder beginnt, diesmal allerdings mit Unterstützung des a-priori Wissens $L_{a,2}^{|}(\mathbf{u}) = L_{e,1}^{-}(\hat{\mathbf{u}})$. Die Eingangsmatrix lautet

$$L_{ch}\mathbf{y} + L_{a,2}^{|}(\mathbf{u}) = \begin{array}{|r|r|r|r|r|}\hline 3{,}1&6{,}9&2{,}1&-2{,}5&6{,}3\\\hline 2{,}6&-1{,}9&0{,}7&1{,}9&-9{,}5\\\hline -8{,}9&4{,}5&-7{,}0&8{,}9&1{,}3\\\hline 3{,}2&-0{,}6&8{,}9&-10{,}2&-12{,}7\\\hline 1{,}9&-5{,}7&7{,}6&-7{,}0&\\\hline\end{array}.$$

Es ist zu beachten, dass die extrinsische Information $L_{e,1}^{|}(\hat{\mathbf{u}})$ von $D^{|}$ der ersten Iteration nicht wieder als a-priori-Wissen an den Eingang von $D^{|}$ zurückgeführt wird. Dies ist sehr wichtig, da sonst eine starke Korrelation mit den übrigen Eingangswerten vorhanden wäre. Es wird also stets **nur der für einen bestimmten Decodierer neue Informationsanteil (die extrinsische Information des anderen Decodierers) weitergeleitet**. Die vertikale Decodierung liefert in der zweiten Iteration jetzt folgendes Ergebnis.

2. Iteration, vertikaler Decoder

$L_{ch}\mathbf{y} + L_{a,2}^{|}(\mathbf{u})$

3,1	6,9	2,1	-2,5	6,3
2,6	-1,9	0,7	1,9	-9,5
-8,9	4,5	-7,0	8,9	1,3
3,2	-0,6	8,9	-10,2	-17,2
1,9	-5,7	7,6	-7,0	

$\Downarrow$

$\mathbf{L}_{e,2}^{|}(\hat{\mathbf{u}})$

-1,9	-0,6	-0,7	1,9	
-1,9	0,6	-2,1	-2,5	
1,9	-0,6	0,7	-1,9	
-1,9	1,9	-0,7	1,9	

$\Longrightarrow$

$\mathbf{L}_1^{|}(\hat{\mathbf{u}}) = L_{ch}\mathbf{y} + L_{a,2}^{|}(\mathbf{y}) + L_{e,2}^{|}(\hat{\mathbf{u}})$

1,2	6,3	1,4	0,6	
0,7	-1,3	-1,4	-0,6	
-7,0	3,9	-6,3	7,0	
1,3	1,3	8,2	-8,3	

2. Iteration, horizontaler Decodierer

Die Eingangswerte des horizontalen Decodierers und das Decodierergebnis lauten in der zweiten Iteration:

$L_{ch}\mathbf{y} + L_{a,2}^{-}(\mathbf{u})$

-1,3	7,0	0,6	-1,3	6,3
3,2	-3,8	1,7	-3,1	-9,5
-5,7	2,6	-5,0	5,7	1,3
-0,6	0,6	7,5	-7,6	-12,7
1,9	-5,7	7,6	-7,0	

$\Longrightarrow$

$\mathbf{L}_{e,2}^{-}(\hat{\mathbf{u}})$

-0,6	0,6	1,3	-0,6	
-1,7	1,7	-3,1	1,7	
-1,3	1,3	-1,3	1,3	
0,6	-0,6	-0,6	0,6	

$\Downarrow$

$\mathbf{L}_{2}^{-}(\hat{\mathbf{u}}) = L_{ch}\mathbf{y} + L_{a,2}^{-}(\mathbf{y}) + L_{e,2}^{-}(\hat{\mathbf{u}})$

-1,9	7,6	2,1	-1,9	
1,5	-2,1	-1,4	-1,4	
-7,0	3,9	-6,3	7,0	
-0,0	0,0	6,9	-7,0	

2. Iteration, geschätzte Daten

$$\hat{\mathbf{u}}_2 = \begin{array}{|c|c|c|c|}\hline 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 \\ \hline x & x & 0 & 1 \\ \hline \end{array}$$

Wie zu sehen ist, sind die zuvor falsch decodierten Binärstellen der ersten Zeile korrigiert worden, dafür treten nun zwei Unsicherheiten in der letzten Zeile auf (1. und 2. Spalte). Die LLR's gleich Null lassen nur eine rein zufällige Entscheidung zu, so dass die Fehlerwahrscheinlichkeit bei diesen beiden Binärstellen bei 50% liegt. In der Hoffnung auf eine Verbesserung dieser beiden Fehler führen wir die Decodierung noch ein drittes Mal durch.

3. Iteration, vertikaler Decodierer

$L_{ch}\mathbf{y} + L_{a,3}^{|}(\mathbf{u})$

0,0	8,2	2,6	-3,8	6,3
3,4	-2,7	0,7	1,1	-9,5
-8,9	4,5	-7,0	8,9	1,3
1,9	-1,9	8,2	-8,9	-12,7
1,9	-5,7	7,6	-7,0	

$\Downarrow$

$\mathbf{L}_{e,3}^{|}(\hat{\mathbf{u}})$

-1,9	-1,9	-0,7	1,1	
0,0	1,9	-2,6	-3,8	
0,0	-1,9	0,7	-1,1	
0,0	2,7	-0,7	1,1	

$\Longrightarrow$

$\mathbf{L}_{3}^{|}(\hat{\mathbf{u}}) = L_{ch}\mathbf{y} + L_{a,3}^{|}(\mathbf{y}) + L_{e,3}^{|}(\hat{\mathbf{u}})$

-1,9	6,3	1,9	-2,7	
3,4	-0,8	-1,9	-2,7	
-8,9	2,6	-6,3	7,8	
1,9	0,8	7,5	-7,8	

3. Iteration, horizontaler Decodierer

$L_{ch}\mathbf{y} + L_{a,3}^{-}(\mathbf{u})$

-1,3	5,7	0,6	-2,1	6,3
5,1	-2,5	1,2	-4,4	-9,5
-7,6	1,3	-5,0	7,5	1,3
1,3	1,4	7,5	-8,4	-12,7
1,9	-5,7	7,6	-7,0	

$\Longrightarrow$

$\mathbf{L}_{e,3}^{-}(\hat{\mathbf{u}})$

-0,6	0,6	1,3	-0,6	
-1,2	1,2	-2,5	1,2	
-1,3	1,3	-1,3	1,3	
1,4	1,3	1,3	-1,3	

$\Downarrow$

$\mathbf{L}_{3}^{-}(\hat{\mathbf{u}}) = L_{ch}\mathbf{y} + L_{a,3}^{-}(\mathbf{y}) + L_{e,3}^{-}(\hat{\mathbf{u}})$

-1,9	6,3	1,9	-2,7	
3,9	-1,3	-1,3	-3,2	
-8,9	2,6	-6,3	8,8	
2,7	2,7	8,8	-9,7	

3. Iteration, geschätzte Daten

$$\hat{\mathbf{u}}_3 = \begin{array}{|c|c|c|c|}\hline 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline \end{array}$$

Nach der dritten Iteration sind nun alle Informationsbit richtig geschätzt worden. Dieses Beispiel soll verdeutlichen, dass bei verketteten Codes die extrinsische Information einer Binärstelle als a-priori-Information für die übrigen Codes eingesetzt werden kann. So konnte in dem obigen Beispiel mit jeder Iteration eine Verbesserung und am Ende sogar eine fehlerfreie Decodierung erzielt werden. Selbstverständlich führt die iterative Decodierung nicht immer zu einer fehlerfreien Lösung.

1.7.6 Generelles Konzept der iterativen Decodierung

Wie im vorangegangenen Abschnitt deutlich wurde, besteht die Grundidee der iterativen Decodierung darin, bei jedem Teildecodierprozeß eine extrinsische Information zu extrahieren, die den nachfolgenden Decodierern als a-priori-Information dienen kann. Es darf jeweils nur diese extrinsische Information weitergereicht werden, da sonst statistische Abhängigkeiten zu den übrigen Eingangswerten eines Decodierers auftreten, die das Decodierergebnis verschlechtern. Bild 1.24 veranschaulicht die prinzipielle Struktur dieses Decodierprozesses für das Beispiel einer parallelen Codeverkettung[2].

Sowohl der horizontale als auch der vertikale Decodierer besitzen zwei Ausgänge: Einer liefert die extrinsische Information $L_e(\hat{\mathbf{u}})$, die zusammen mit den Kanalausgangswerten das Eingangssignal eines Teildecodierers bildet. Der zweite Ausgang enthält das komplette Decodierergebnis $L(\hat{\mathbf{u}})$, aus dem dann durch Hard-Decision die Decodierentscheidung gewonnen werden kann.

Der Name 'Turbo-Codes'
Für die in Abschnitt 1.4.2 vorgestellten Turbo-Codes ist Bild 1.24 leicht zu modifizieren, um das Interleaving explizit sichtbar zu machen. Ferner ist Gl. (1.41) zu entnehmen ist, dass die extrinsische Information durch

[2]Bei einer seriellen Codeverkettung sind die Codesymbole $\mathbf{c}_1$ des äußeren Codierers C_1 die Eingangssymbole des inneren Codierers C_2. Daher muss der äußere Decodierer D_2 Soft-Werte für die codierten Bit $\mathbf{c}_1$ liefern, nicht nur für die Informationsbit $\mathbf{u}$.

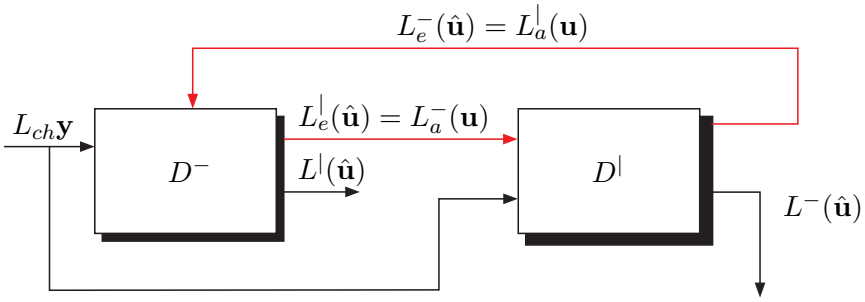

Bild 1.24: Iterativer Decodierprozeß

Subtraktion

$$L_e(\hat{u}_i) = L(\hat{u}_i) - L_{ch}y_i - L_a(u_i) \tag{1.62}$$

extrahiert werden kann. Wir erhalten somit die in Bild 1.25 skizzierte Struktur.

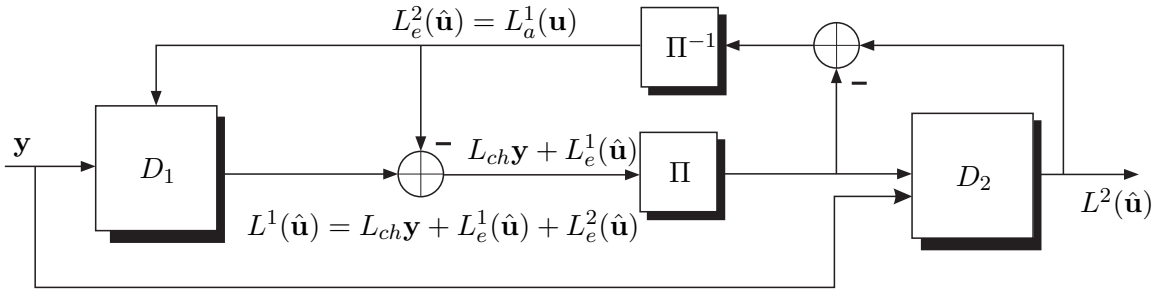

Bild 1.25: Iterativer Decodierprozeß für Turbo-Codes

Die von den Decodierern erzeugten *Soft*-Werte $L^1(\hat{\mathbf{u}})$ bzw. $L^2(\hat{\mathbf{u}})$ setzten sich bekanntermaßen aus drei Anteilen zusammen, von denen der Anteil der a-priori-Information $L_a(\hat{\mathbf{u}})$ subtrahiert wird. Übrig bleiben der direkte Informationsanteil der Kanalausgangswerte $L_{ch}\mathbf{y}$ und die extrinsische Information $L_e(\hat{\mathbf{u}})$, wobei letztere für den nachfolgenden Decodierer eine a-priori-Information bildet.

1.7.7 Ergebnisse zur iterativen Decodierung

Im letzten Abschnitt sollen noch einmal ein paar exemplarische Ergebnisse zur iterativen Decodierung vorgestellt werden, um die recht theoretischen Herleitungen der letzten Abschnitte zu veranschaulichen. Außerdem kann die Abhängigkeit der Leistungsfähigkeit verketteter Codes von wichtigen Parametern verdeutlicht werden.

Die Bilder 1.26 und 1.27 zeigen die Ergebnisse der *Union-Bound*-Abschätzung für drei verschiedene Produktcodes. Sie wurden aus drei Hamming-Codes konstruiert, dem (7,4)-Hamming-Code, dem (15,11)-Hamming-Code und dem (31,26)-Hamming-Code. Dabei wurde nur die Mindestdistanz $d_{\min} = 5$ berücksichtigt. Sie ist für alle drei Codes identisch, da Hamming-Codes stets die Mindestdistanz 3 besitzen und somit der unvollständige Produktcode die Mindestdistanz $3 + 3 - 1 = 5$. Lediglich der Koeffizient c_5 aus Gl. (1.11) ist unterschiedlich, weshalb die Verläufe sich leicht unterscheiden ((7,4): $c_5 = 0,5625$, (15,11): $c_5 = 0,2975$, (31,26): $c_5 = 0,1479$). Asymptotisch spielt dieser Faktor keine Rolle mehr.

Die Kurven sind also nur für große Signal-Rausch-Abstände genau. Trotzdem sind sie geeignet, um einige tendenzielle Aussagen zu treffen. Trägt man die Kurven über E_s/N_0 auf, ergeben sich wegen

$$P_b \leq c_5 \cdot \operatorname{erfc}\left(\sqrt{5\frac{E_s}{N_0}}\right)$$

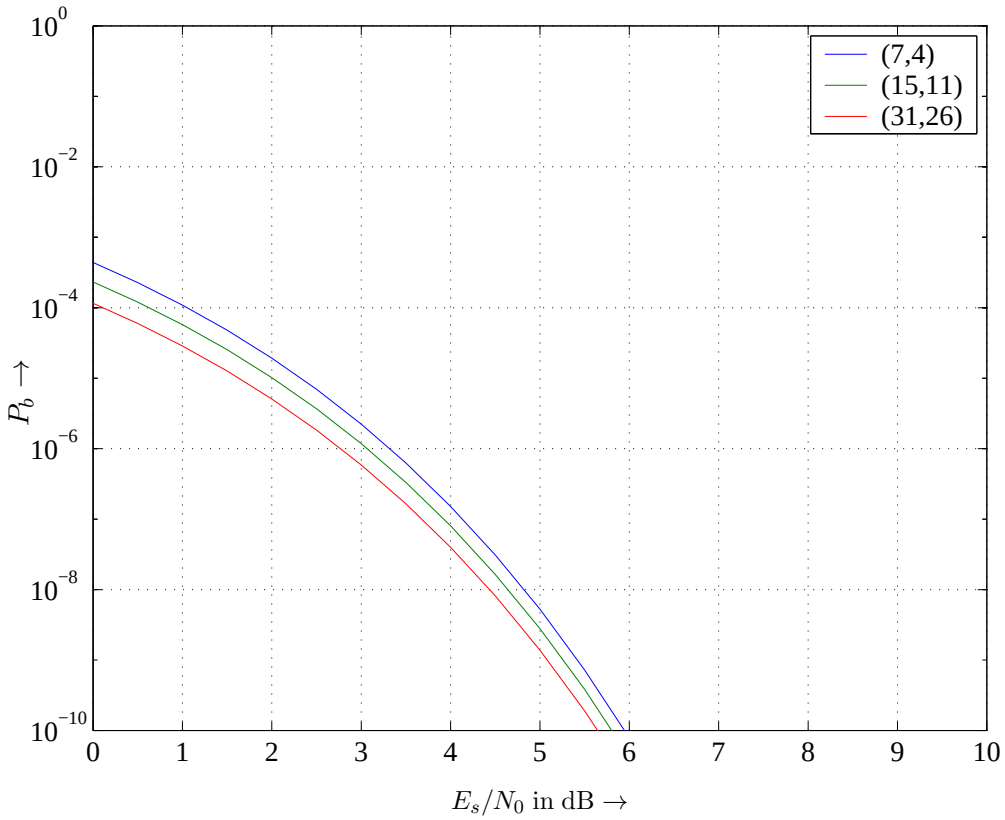

Bild 1.26: *Union-Bound*-Abschätzung (asymptotisch) für Produktcodes a) über E_s/N_0 und b) über E_b/N_0

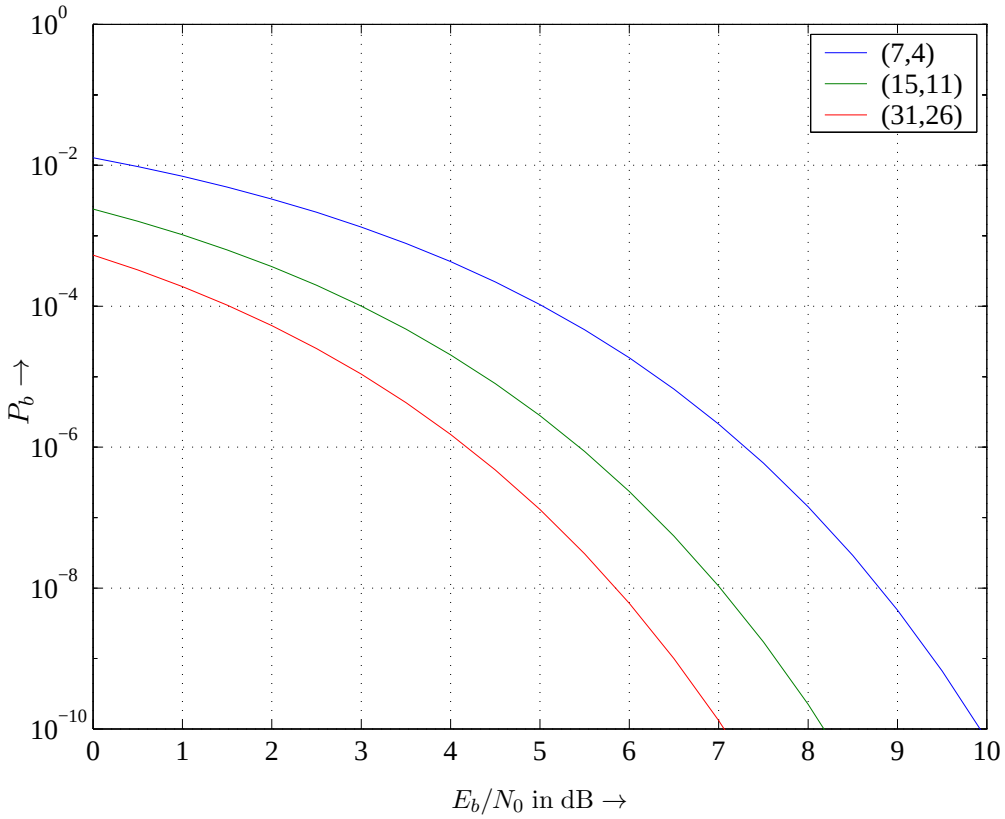

Bild 1.27: *Union-Bound*-Abschätzung (asymptotisch) für Produktcodes über E_b/N_0

nahezu keine Unterschiede, was aufgrund der gleichen Mindestdistanz nicht verwunderlich ist. Allerdings ist zu beachten, dass die drei Codes jeweils unterschiedliche Coderaten besitzen und somit auch die erforderliche

Bandbreite variiert. Berücksichtigt man diese Tatsache und trägt über E_b/N_0 auf,

$$P_b \leq c_5 \cdot \mathrm{erfc}\left(\sqrt{5\frac{R_c E_b}{N_0}}\right)$$

so erhält man die in Bild (1.27) dargestellten Verläufe. Es wird der deutliche Vorteil des (31,26)-Hamming-Codes deutlich, da er einerseits den geringsten Koeffizienten c_5 besitzt, andererseits die Distanz $d_{\min} = 5$ mit wesentlich geringerer Redundanz erreicht und somit eine größere spektrale Effizienz besitzt. Bezogen auf die je Informationsbit übertragene Energie ist dieser Produktcode also der beste der drei Kandidaten.

In den folgenden drei Bildern (1.28 - 1.30) sind Simulationsergebnisse dargestellt. Es wurden insgesamt stets 3 Decodierdurchläufe durchgeführt. Zum Vergleich ist jeweils das Ergebnis der analytischen Abschätzung ebenfalls mit aufgeführt.

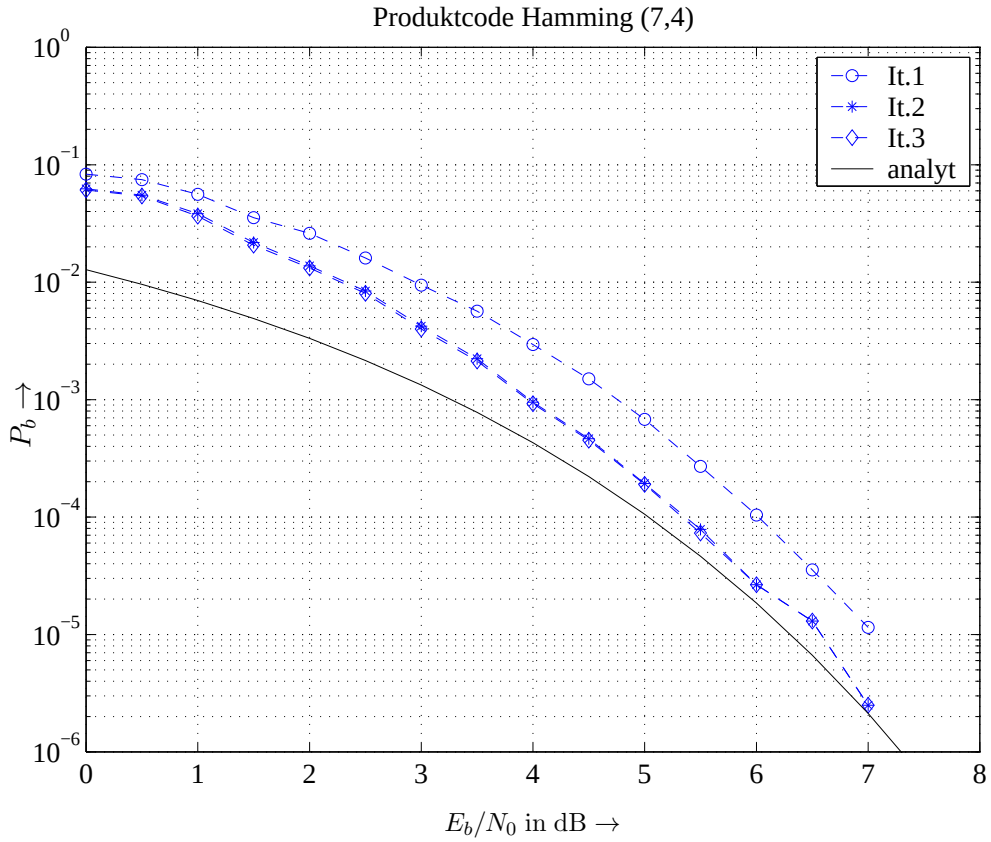

Bild 1.28: Bitfehlerraten für Produktcode aus zwei (7,4)-Hamming-Codes

Es ist erkennbar, dass mit jeder Iteration der zusätzliche Gewinn kleiner ausfällt. Dies liegt an der Tatsache, dass stets die gleichen Informationsbit geschätzt werden und die als a-priori-Information genutzte extrinsische Information immer stärker mit den übrigen Eingangsdaten korreliert. (Dies verdeutlicht noch einmal die Wichtigkeit der statistischen Unabhängigkeit der a-priori-Information von den übrigen Eingangssignalen). Die Gewinne zusätzlicher Iterationen fallen um so höher aus, je größer der Interleaver ist. Je weiter benachbarte Bit durch ihn auseinander gewürfelt werden, desto besser ist ihre statistische Unabhängigkeit erfüllt.

In Bild 1.30 ist außerdem deutlich zu erkennen, dass die Bitfehlerkurven ab einem gewissen Signal-Rausch-Abstand abflachen. Dies ist nicht auf Simulationsungenauigkeiten zurückzuführen, wie der Vergleich mit dem asymptotischen Verlauf der analytischen Abschätzung zeigt. Für große Signal-Rausch-Abstände dominiert einzig und allein die Minimaldistanz, und diese bestimmt die Steigung der Fehlerkurven. In Bild 1.29 is dieser Effekt ebenfalls zu beobachten, allerdings nicht so ausgeprägt.

Die Bilder 1.32 und 1.31 zeigen noch einmal den Vergleich der drei Produktcodes, nun aber anhand von Simulationsergebnissen. Über E_s/N_0 aufgetragen ist zu erkennen, dass die Mindestdistanz für große Signal-Rausch-

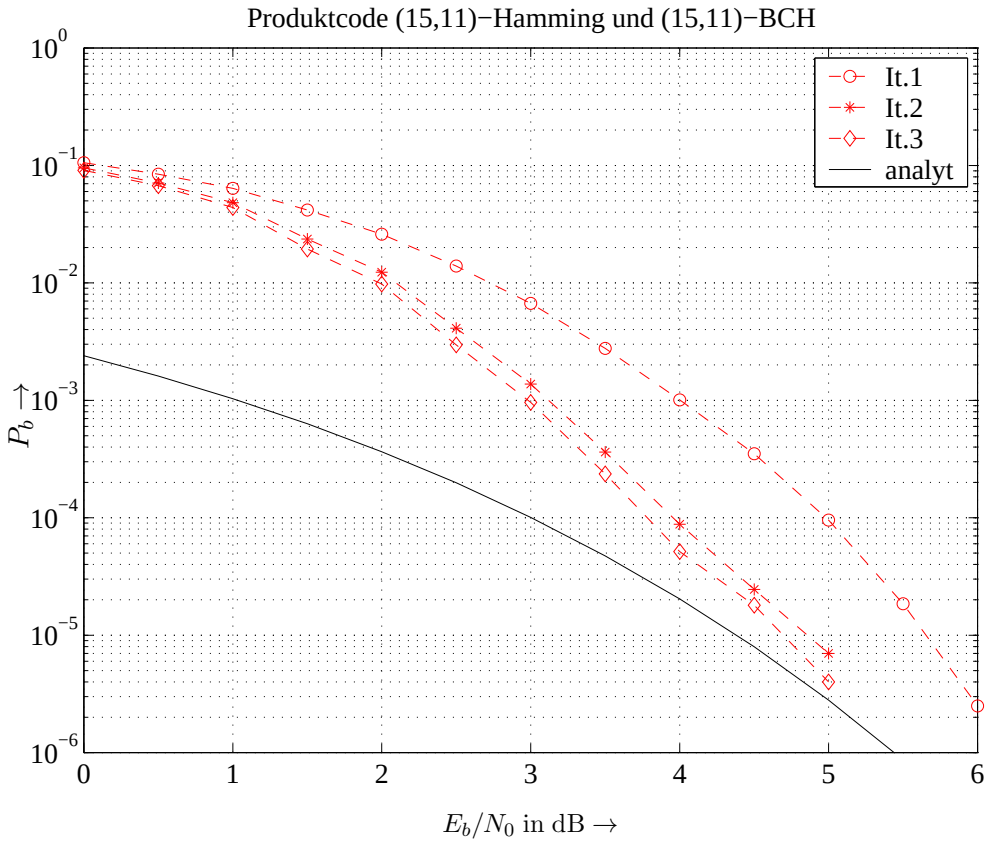

Bild 1.29: Bitfehlerraten für Produktcode aus zwei (15,11)-Hamming-Codes

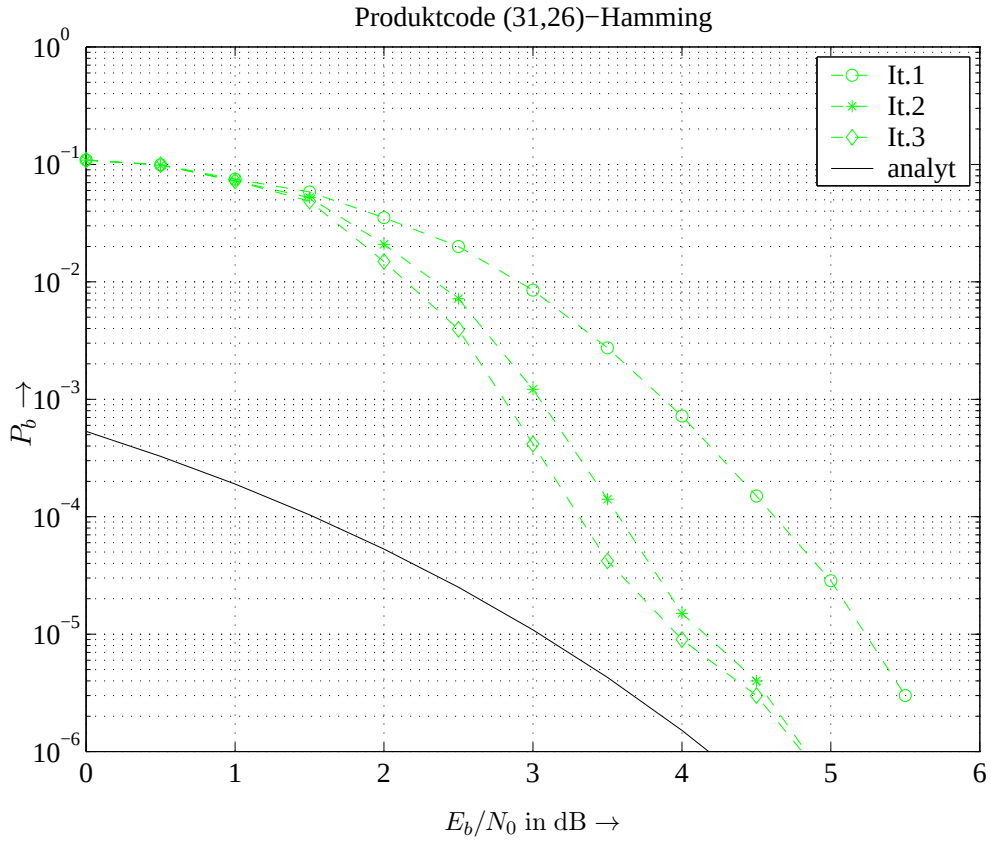

Bild 1.30: Bitfehlerraten für Produktcode aus zwei (31,26)-Hamming-Codes

Abstände dominiert, hier verlaufen alle drei Kurven sehr dicht beieinander. Für kleinere SNR erweist sich der (7,4)-Hamming-Code als der beste Komponentencode.

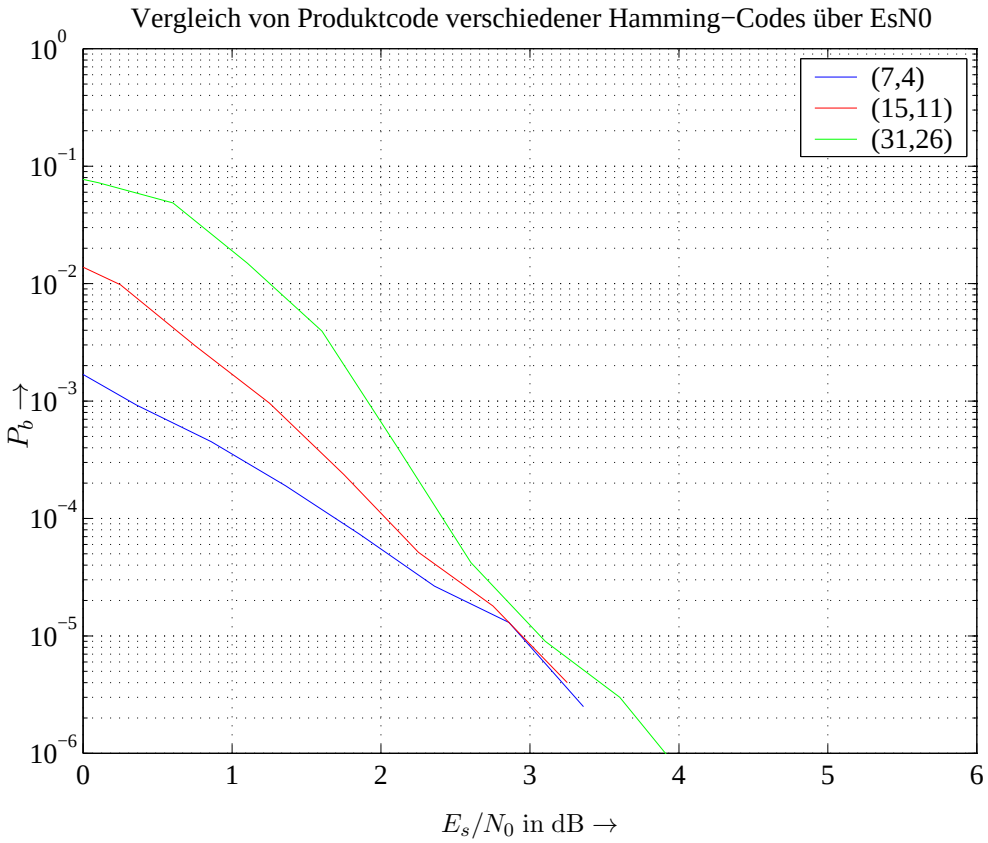

Bild 1.31: Vergleich der Bitfehlerraten für Produktcodes über E_s/N_0

Berücksichtigt man wiederum die unterschiedlichen Coderaten, so ergeben sich die in Bild 1.32 dargestellten Verhältnisse. Aufgrund der besten spektralen Effizienz besitzt der (31,26)-Hamming-Code die größte Leistungsfähigkeit.

Zum Ende dieses Kapitels zeigen die Bilder 1.33 bis 1.37 die Ergebnisse für die in Abschnitt 1.4.2 vorgestellten Turbo-Codes. Es wird deutlich, dass auch hier mit zunehmender Iterationszahl die Gewinne immer kleiner werden. Weiterhin führt eine Vergrößerung des Interleavers zu einer Verringerung der Bitfehlerrate. Allerdings sind hier lediglich einfache Blockinterleaver eingesetzt worden. Wie schon diskutiert wurde, sind aber pseudozufällige Interleaver besser geeignet. Dies ist daran zu erkennen, dass eine Vergrößerung von 400 auf 900 Bit keinen großen Gewinn mehr liefert.

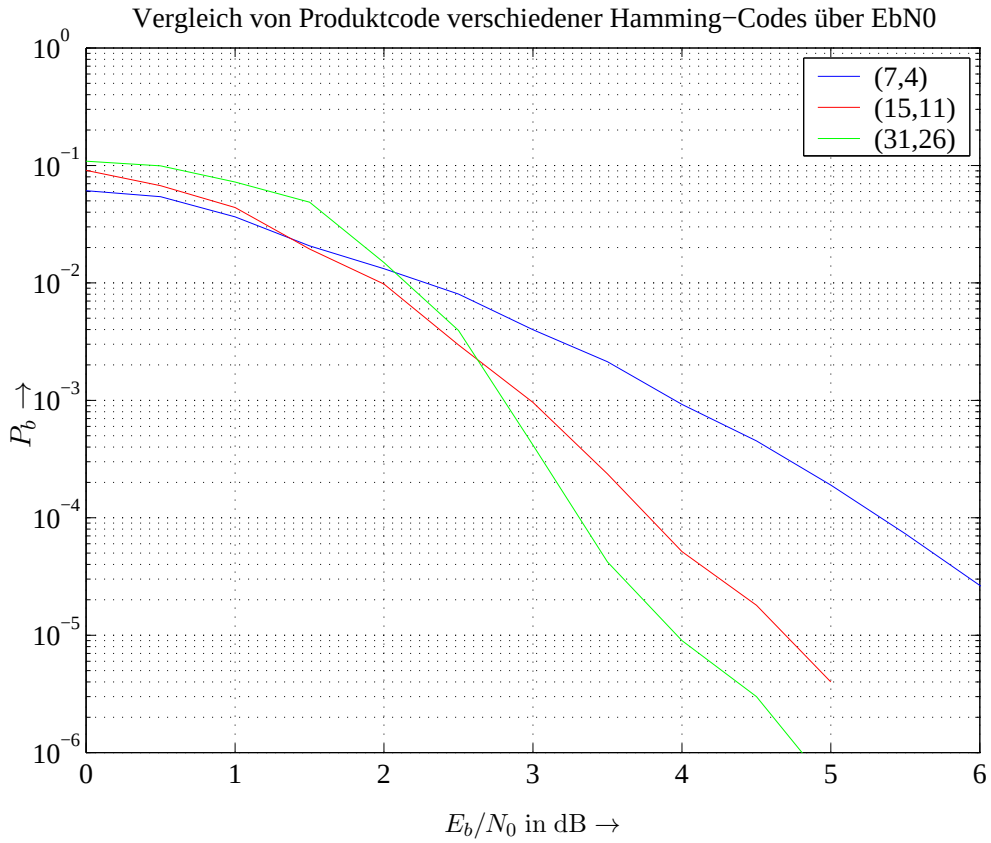

Bild 1.32: Vergleich der Bitfehlerraten für Produktcodes über E_b/N_0

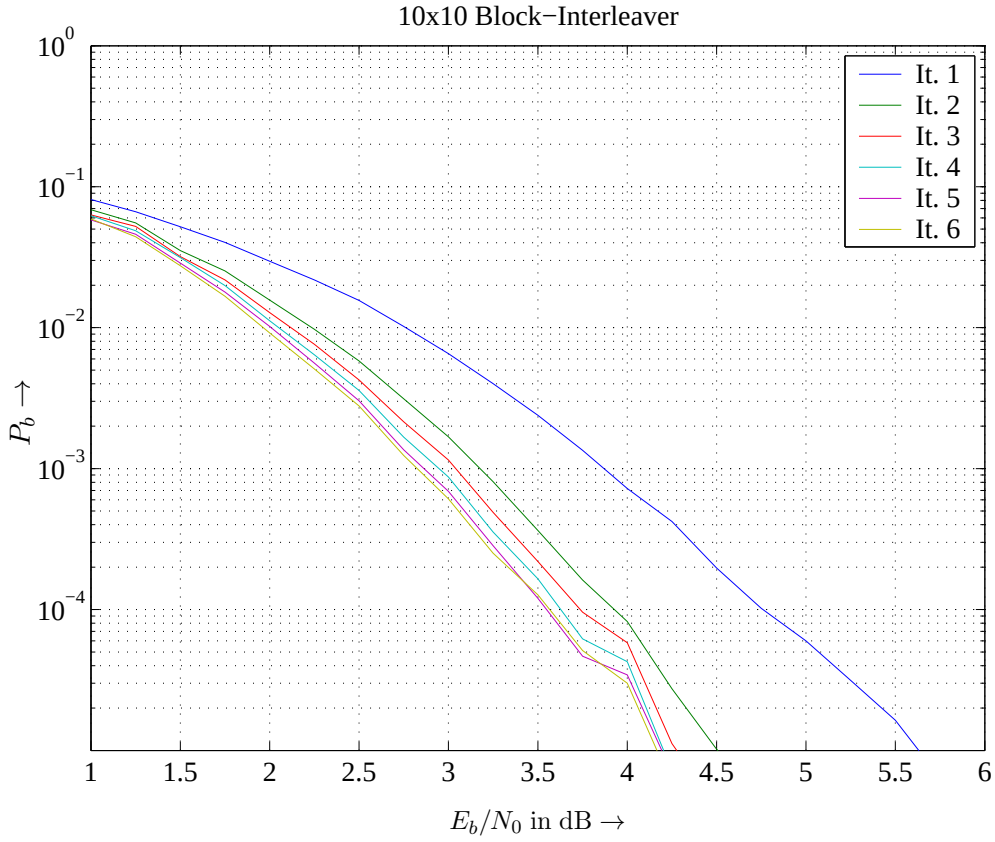

Bild 1.33: Bitfehlerraten für Turbo-Code mit 10x10-Blockinterleaver

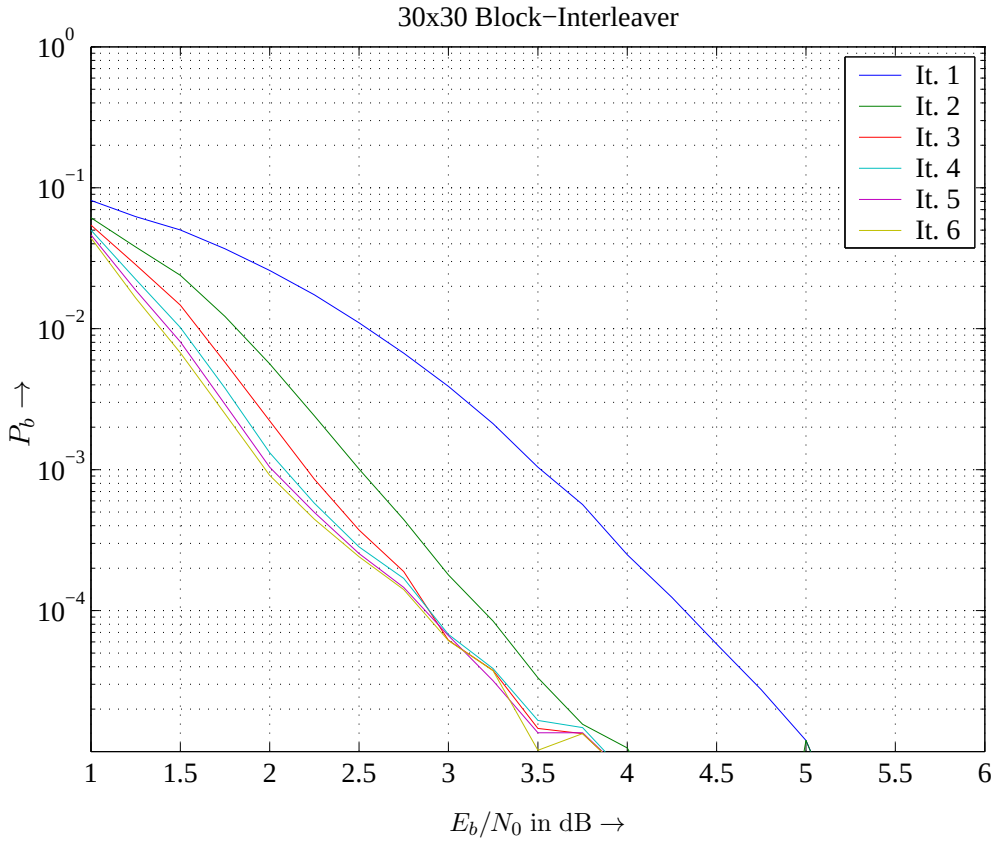

Bild 1.34: Bitfehlerraten für Turbo-Code mit 30x30-Blockinterleaver

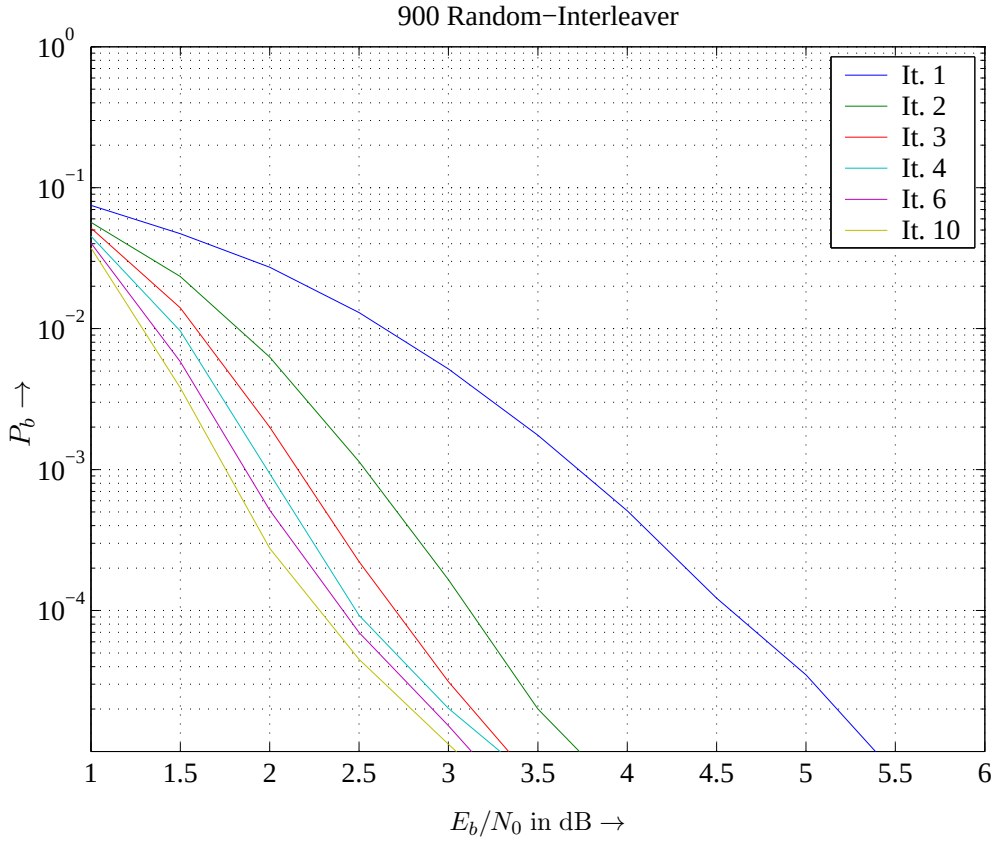

Bild 1.35: Bitfehlerraten für Turbo-Code mit 30x30-Zufallsinterleaver

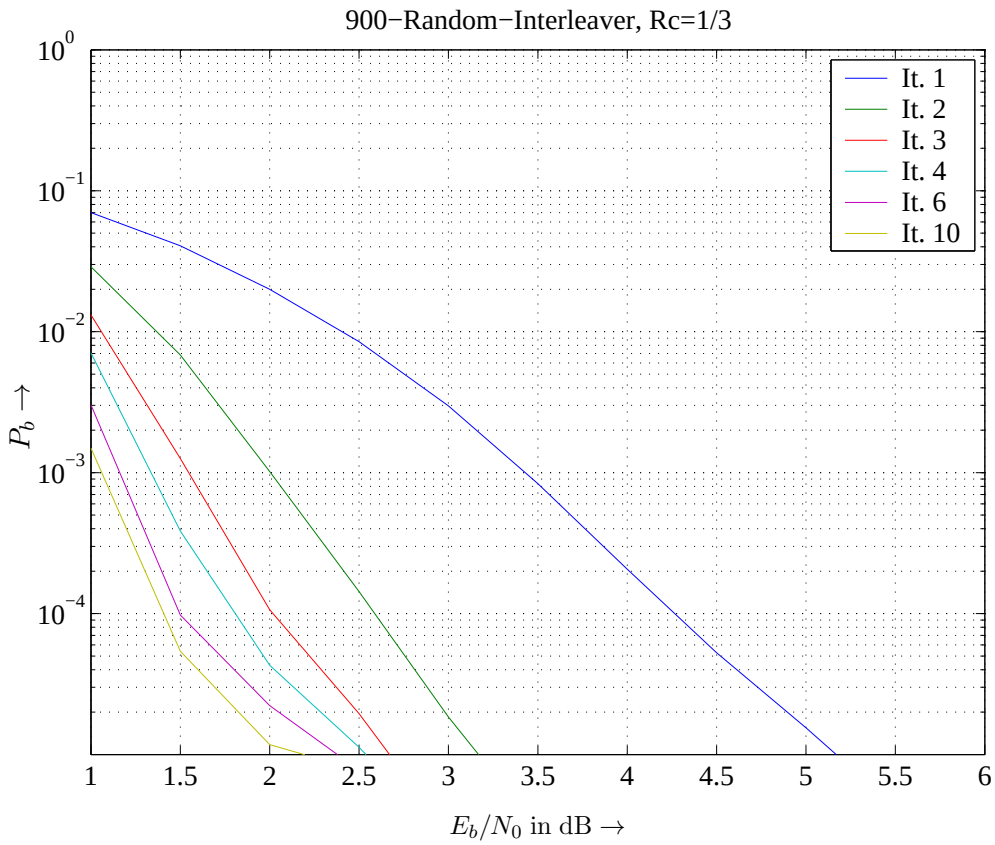

Bild 1.36: Vergleich der Bitfehlerraten für Turbo-Code mit 30x30-Zufallsinterleaver und $R_c = 1/3$

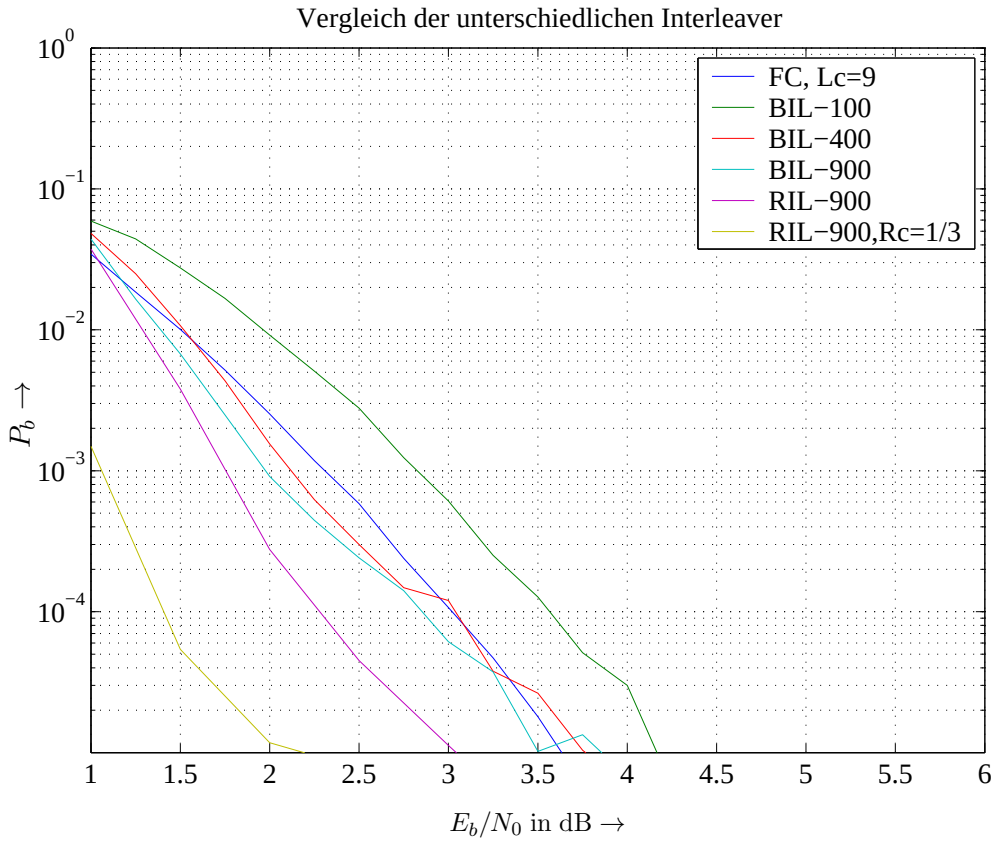

Bild 1.37: Vergleich der Bitfehlerraten für Turbo-Code mit verschiedenen Interleavergrößen ($R_c = 1/2$)