Partitionierung von Faltungscodes zur verallgemeinerten Verkettung

Diplomarbeit

von

Günther Haas

Abteilung Informationstechnik
Universität Ulm
Januar 1997

D/1996/AH/3

Abteilung Informationstechnik Universität Ulm

DIPLOMARBEIT

Partitionierung von Faltungscodes zur verallgemeinerten Verkettung

Erläuterungen:

Bei der verallgemeinerten Verkettung von Codes wird ein innerer Code partitioniert und mit mehreren äußeren Codes verkettet. Dieses sonst für Blockcodes gängige Prinzip soll in der Arbeit auf gewöhnliche Faltungscodes angewandt werden. Ein wichtiger Punkt hierbei ist die Partitionierung des inneren Faltungscodes, denn die sich dabei ergebenden Untercodes sollten eine ansteigende freie Distanz haben. Um geeignete Untercodes zu erhalten hat es sich kürzlich gezeigt, daß dazu die Konstruktion eines Verwürflers (Scramblers) notwendig ist. Durch dessen Verwendung werden äquivalente Faltungscodes konstruiert, die bessere Distanzeigenschaften besitzen als die gewöhnlichen Faltungscodes. Die Untercodes des äquivalenten inneren Faltungscodes können somit mit äußeren Faltungscodes unter Berücksichtigung eines Zwischeninterleavers verkettet werden. Es ist noch zu untersuchen, welche Performance (Coderate / Bitfehlerrate) sich durch diese Codekonstruktionen erzielen läßt.

In dieser Arbeit sollen bekannte Faltungscodes partitioniert werden. Dabei sind jeweils optimale Scramblermatrizen und somit äquivalente Faltungscodes zu bestimmen. Diese sind mit geeigneten äußeren Codes zu verketten. Ein zum Teil vorhandenes Programmpaket (`C++`) kann dabei benutzt und erweitert werden. Weiterhin sollen Simulationen zur Untersuchung der Leistungsfähigkeit der Partitionierung bzw. der Codekonstruktion durchgeführt werden. Das Simulationsprogramm ist dabei für `COSSAP` (Programmiersprache `C`) zu erstellen bzw. zu erweitern.

Abgabetermin:	30. Januar 1997
Bearbeiter:	Günther Haas
Betreuer:	Dipl.-Ing. Armin Häutle, Universität Ulm Dipl.-Ing. Hans Dieterich, AEG Mobile Communications, Ulm
Prüfer:	Prof. Dr.-Ing M. Bossert
Katalognr.:	D/1996/AH/3

Ich versichere, daß ich die vorliegende Diplomarbeit selbständig und ohne unzulässige fremde Hilfe angefertigt habe.

Ulm, den 30. Januar 1997

(Günther Haas)

Inhaltsverzeichnis

KAPITEL

1

Einleitung

Heutzutage wird bei nahezu allen digitalen Übertragungsverfahren Kanalcodierung eingesetzt, um die zu übertragenden Nachrichten gegen Störungen auf dem Kanal zu schützen. Auch in anderen Bereichen, etwa bei der Speicherung von digitalen Daten, werden inzwischen Codierungsverfahren verwendet.

Es wurde und wird daher immer wieder nach neuen, verbesserten Codier- und Decodiermethoden gesucht. Bei dieser Suche kam schon bald die Idee auf, durch die Verkettung bekannter Codes neue, längere Codes zu konstruieren. Erste Ideen zur seriellen Verkettung, also zur aufeinanderfolgenden Anwendung zweier Codes, wurden bereits 1966 von Forney entwickelt. In den folgenden Jahren wurde die Codeverkettung von verschiedenen russischen Wissenschaftlern weiterentwickelt. 1974 wurden von Blokh und Zyablov erstmals verallgemeinert verkettete Codes vorgestellt [BZ74] und 1976 wurde schließlich von Zinoviev das mächtige Konzept der *verallgemeinerten Codeverkettung* (engl. generalized concatenation, GC) beschrieben. Es läßt sich im wesentlichen wie folgt zusammenfassen:

- Ein innerer Code wird partitioniert, das heißt aufgeteilt in Untercodes, und die Untercodes werden numeriert.
- Die Numerierung der inneren Codes wird durch äußere Codes geschützt.

Dieses Prinzip wurde bis heute in den meisten Fällen zur Verkettung von Blockcodes angewandt. In den letzten Jahren gab es einige wenige Ansätze zur verallgemeinerten Verkettung von Faltungscodes (z.B. von Zyablov und Shavgulidze im Jahre 1991, [ZS91]).
Insbesondere die wesentliche Frage der Partitionierung eines Faltungscodes wurde bis heute nicht befriedigend geklärt. Grundlegende Ideen zur Partitionierung von Faltungscodes für die verallgemeinerte Codeverkettung wurden erst 1996 von Bossert, Dieterich und Shavgulidze in [BDS96a] und [BDS96b] vorgestellt. Sie basieren auf der Tatsache, daß für jeden Faltungscode verschiedene äquivalente Generatormatrizen bzw. Encoder existieren. Das Einfügen eines sogenannten Scramblers ändert daher nicht die Menge der Codesequenzen, ermöglicht aber eine gezielte Partitionierung eines Faltungscodes. Wichtigstes Ziel dieser Partitionierung ist es, Untercodes mit ansteigenden freien Distanzen zu erhalten. Die Decodierkomplexität für den inneren Code soll dabei durch den Scrambler möglichst nicht erhöht werden.

Die vorliegende Arbeit beschäftigt sich mit der Untersuchung und Weiterentwicklung der Partitionierungsverfahren aus [BDS96a] und [BDS96b], sowie mit der Frage nach einem geeigneten Decodierverfahren für die inneren Untercodes bei verallgemeinerter Verkettung.

Außerdem wird eine Methode zur Konstruktion eines verallgemeinert verketteten Codes (*GC-Codes*) beschrieben und die Performance solcher Codes wird untersucht.

In Kapitel 2 werden zunächst verschiedene Darstellungsformen für Faltungscodes beschrieben, welche zur Beschreibung der Scrambler und Codes in den folgenden Kapiteln benötigt werden. Besonders wichtig sind dabei die Matrix- und Polynomschreibweise sowie die Beschreibung eines Faltungscodes als Unit Memory Code und seine Darstellung im Codetrellis.

In Kapitel 3 wird die *zufällige Partitionierung* eines Faltungscodes beschrieben, die sich ohne Verwendung eines Scramblers ergibt. Danach wird die Konstruktion und Funktionsweise sogenannter Block- und Diagonalscrambler zur gezielten Partitionierung erläutert. Dabei wird neben dem Verfahren der *Zustandspunktierung* nach [BDS96a] und [BDS96b] die Partitionierung von Faltungscodes durch *Übergangspunktierung* im Trellis vorgestellt.
Dieses Verfahren wird verallgemeinert und es werden sogenannte *Unit Memory Scrambler* und *Faltungsscrambler* vorgestellt.
Zum Abschluß des Kapitels wird die Anwendung der Partitionierungsverfahren auf punktierte Codes erweitert und ein Suchalgorithmus für Scrambler angegeben.

Ein wesentlicher Gesichtspunkt bei Verwendung von Kanalcodierung in der Praxis ist die Decodierkomplexität. Diese soll durch den Einsatz eines Scramblers nach Kapitel 3 nicht erhöht werden. Außerdem benötigt man ein geeignetes Decodierverfahren zur Decodierung der Untercodes des partitionierten Faltungscodes.
In Kapitel 4 wird deshalb zuerst das Decodierprinzip bei verallgemeinerter Codeverkettung erläutert. Anschließend wird ein bekannter Decodieralgorithmus, der MAP-Algorithmus nach Bahl u. a., so modifiziert, daß er alle gestellten Forderungen erfüllt.

In Kapitel 5 wird anhand eines Beispiels die verallgemeinerte Verkettung von Faltungscodes betrachtet. Hierbei werden zunächst die Untercodes, die sich mit Hilfe eines Scramblers ergeben, im Hinblick auf die erreichbare Bitfehlerrate untersucht. Darauf aufbauend wird in Abschnitt 5.2 eine mögliche Methode zur Auswahl geeigneter äußerer Codes angegeben. Die so konstruierten GC-Codes werden in Abschnitt 5.3 und in Abschnitt 5.4 mit Hilfe von Abschätzungen und Simulationen auf ihre Leistungsfähigkeit hin untersucht. Insbesondere wird dabei die Wirkung des Scramblers betrachtet.

Abschließend werden die Ergebnisse der vorliegenden Arbeit in Kapitel 6 zusammengefaßt, und es wird ein Überblick über mögliche Anwendungen und denkbare Weiterentwicklungen gegeben.

KAPITEL

2

Grundlagen und Definitionen

2.1 Faltungscodes

Im folgenden werden verschiedene Schreibweisen für Faltungscodes in dieser Arbeit festgelegt. Sie orientieren sich in erster Linie an [JW93], [BDS96a] und [BDS96b], weichen jedoch in manchen Punkten davon ab, wenn eine andere Schreibweise geeigneter erscheint.

Man kann einen Faltungscode auf mehrere gleichwertige Weisen beschreiben. In allen Darstellungen bleibt jedoch die Coderate $r = k/n$ unverändert.

2.1.1 Verschiedene Darstellungsformen

Realisierung als Schieberegister, Codeparameter

Die Realisierung (den Encoder) eines Faltungscodes der Rate $r = k/n$ kann man als k binäre Schieberegister[1] darstellen, deren Speicherinhalte auf n verschiedene Weisen verknüpft werden. Die resultierenden n Werte werden an n verschiedene Ausgänge geleitet. Diese sogenannte „*Controller Canonical Form*" (CCF, siehe [JW93]) ist in Bild 2.1 für einen Faltungscode mit $k = 2$ und $n = 3$ dargestellt. Die im Bild angegebenen Bezeichnungen werden im weiteren Verlauf dieses Kapitels noch erläutert.
Nach dieser Darstellung kann man einen Code der Dimension $k > 1$ auch auffassen als k Codes der Rate $1/n$, mit denen k unabhängige Informationsbitströme codiert werden und deren Codebitströme anschließend modulo 2 aufaddiert werden. Eine Informationsbitfolge $\underline{i}$ am Eingang des Encoders hat einen Codebitstrom $\underline{c}$ am Ausgang zur Folge. Für die mathematische Beschreibung ist es sinnvoll, diese Bitfolgen als halbunendliche Vektoren zu betrachten:

$$\text{Informationsfolge}: \quad \underline{i} = (i_0, i_1, i_2, \ldots, i_l, \ldots), \qquad i_l = 0 \text{ für } l < 0 \tag{2.1}$$

$$\text{Codebits}: \quad \underline{c} = (c_0, c_1, c_2, \ldots, c_j, \ldots), \qquad c_j = 0 \text{ für } j < 0 \tag{2.2}$$

Wichtige Parameter eines Faltungscodes lassen sich direkt aus der Realisierung des Encoders nach Bild 2.1 ablesen: Die Anzahl der Speicherelemente des i-ten Schieberegisters bezeichnet man als *Einflußlänge* ν_i (engl.: constraint length).

[1] siehe hierzu auch [Bos94]: FIR-Systeme, Transversalfilter

Die Summe aller Speicherelemente des Encoders in Controller Canonical Form wird *Gesamteinflußlänge* ν (engl.: overall constraint length) genannt und berechnet sich zu

$$\nu = \sum_{i=1}^{k} \nu_i. \tag{2.3}$$

Schließlich definiert man als *Gedächtnislänge* m (engl.: memory) die Anzahl Speicherelemente des längsten der insgesamt k Eingangsschieberegister:

$$m = \max_i \{\nu_i\} \tag{2.4}$$

Beispiel 2.1:
Für den Encoder nach Bild 2.1 sind die Einflußlängen der Eingangsschieberegister $\nu_1 = \nu_2 = 2$. Daraus ergibt sich die Gedächtnislänge $m = 2$ und die Gesamteinflußlänge $\nu = 2 + 2 = 4$.

□

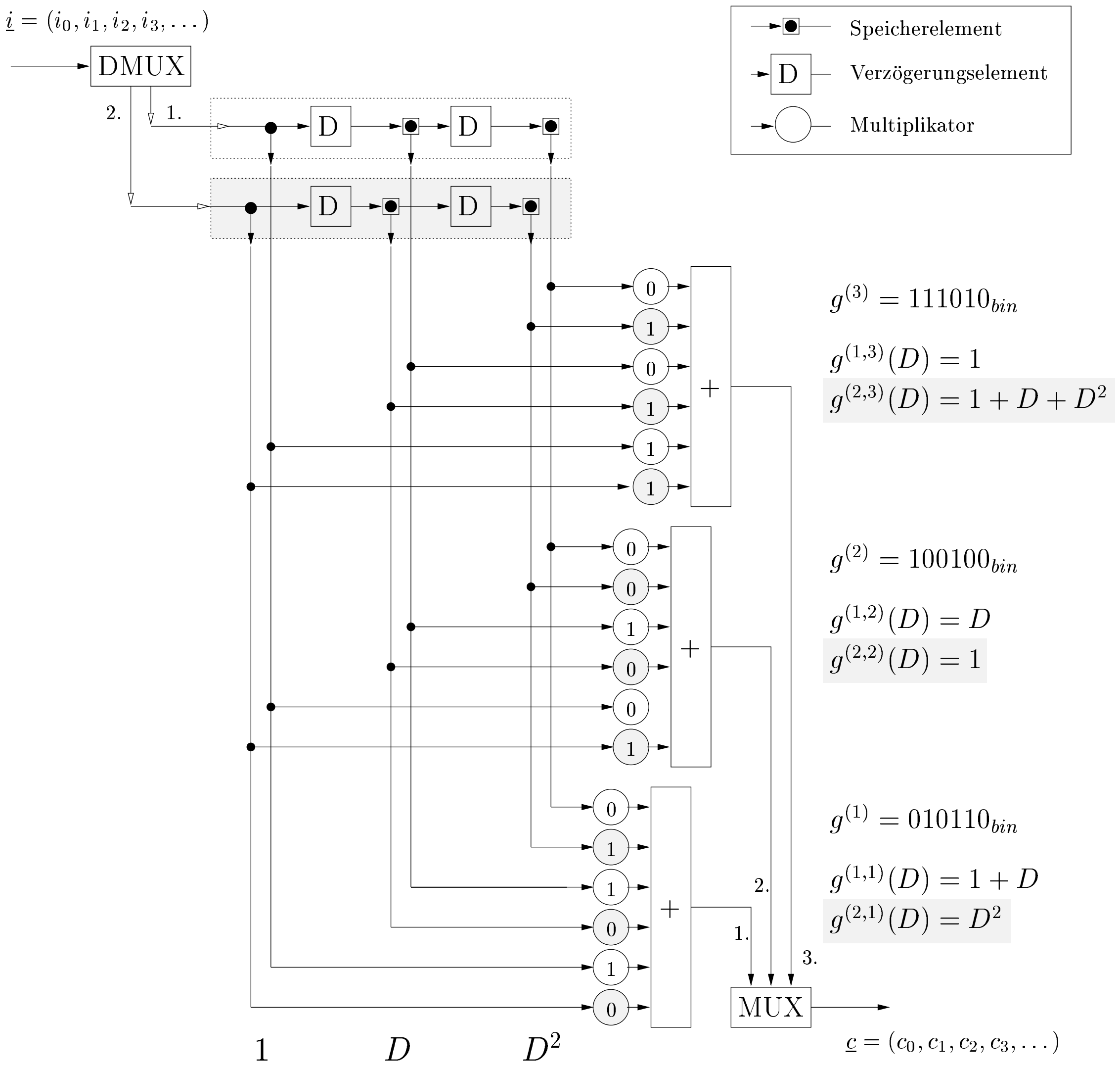


Abbildung 2.1: Encoder des Faltungscodes (26, 44, 72) in Controller Canonical Form

Ein weiterer wichtiger Parameter eines Faltungscodes ist seine *freie Distanz* d. Man bezeichnet damit die kleinste vorkommende Hamming-Distanz (siehe [Bos92], [Haa96]) zwischen zwei Codesequenzen, äquivalent zur Mindestdistanz bei Blockcodes . Bei linearen Codes ist diese gleich dem kleinsten Hamming-Gewicht einer Codefolge $\underline{c} \neq \underline{0}$.
Die freie Distanz ist ein erstes Kriterium für die Korrekturfähigkeit eines Faltungscodes.

Ein Faltungscode $\mathcal{C}$ der Rate $r = k/n$ mit der freien Distanz d wird im allgemeinen mit $\mathcal{C}(n, k, d)$ bezeichnet.

Oktaldarstellung

Zur Beschreibung von Faltungscodes wird meist die sogenannte Oktaldarstellung verwendet, wie sie in [Pro89] (S. 444) beschrieben ist. Die zugehörige Binärdarstellung läßt sich direkt aus der Realisierung als Schieberegister in CCF ablesen.

Beispiel 2.2: Für den Faltungscode nach Bild 2.1 erhält man, wenn man die binären Filterkoeffizienten vor den Addierern von unten nach oben ausliest, folgende Darstellung:

$$\begin{aligned} g^{(1)} &= 010110_{\text{bin}} = 26_{\text{okt}} \\ g^{(2)} &= 100100_{\text{bin}} = 44_{\text{okt}} \\ g^{(3)} &= 111010_{\text{bin}} = 72_{\text{okt}} \end{aligned}$$

□

Die Werte $g^{(i)}$ werden auch als *oktale Generatorpolynome* bezeichnet und allgemein in folgender Form angegeben:

$$(g^{(1)}, g^{(2)}, \dots, g^{(n)}) \tag{2.5}$$

Matrixdarstellung

Der Zusammenhang zwischen Informations- und Codefolge läßt sich durch eine Matrixmultiplikation beschreiben:

$$\underline{c} = \underline{i} \cdot \underline{G} \tag{2.6}$$

Die Matrix $\underline{G}$ wird als *Generatormatrix* bezeichnet. Es handelt sich dabei um eine halbunendliche Matrix, deren Koeffizienten sich periodisch wiederholen. Diese Koeffizienten entsprechen der Binärdarstellung der oktalen Generatorpolynome (s.o), also den Koeffizienten im Schieberegister, und sind in Beispiel 2.3 fett hervorgehoben.

Beispiel 2.3: Für den Faltungscode aus Beispiel 2.2 und Bild 2.1 ergibt sich die Generatormatrix

$$\underline{G} = \begin{bmatrix} 101 & 110 & \mathbf{000} & 000 & 000 & \dots \\ 011 & 001 & \mathbf{101} & 000 & 000 & \dots \\ \\ 000 & 101 & \mathbf{110} & 000 & 000 & \dots \\ 000 & 011 & \mathbf{001} & 101 & 000 & \dots \\ \\ 000 & 000 & \mathbf{101} & 110 & 000 & \dots \\ 000 & 000 & \mathbf{011} & 001 & 101 & \dots \\ \\ 000 & 000 & 000 & 101 & 110 & \dots \\ 000 & 000 & 000 & 011 & 001 & \dots \\ & & \vdots & & & \ddots \end{bmatrix}$$

□

Entsprechend der Anzahl der Eingangsschieberegister bzw. der Verknüpfungen in Bild 2.1 kann man je k Eingangsbits und n Codebits zusammenfassen:

$$\begin{aligned} \text{Informationsfolge}: \quad & \underline{u} = (\underline{u}_0, \underline{u}_1, \underline{u}_2, \ldots, \underline{u}_t, \ldots), \quad && \underline{u}_t = \underline{0} \text{ für } t < 0 && (2.7)\\ \text{Codefolge}: \quad & \underline{v} = (\underline{v}_0, \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_t, \ldots), \quad && \underline{v}_t = \underline{0} \text{ für } t < 0 && (2.8) \end{aligned}$$

Dabei stellen die Elemente der Informationsfolge $\underline{u}$ Zeilenvektoren mit k Komponenten dar, die Elemente der Codefolge $\underline{v}$ sind Zeilenvektoren mit n Komponenten:

$$\underline{u}_t = (u_t^{(1)}, \ldots, u_t^{(k)}) \tag{2.9}$$

$$\underline{v}_t = (v_t^{(1)}, \ldots, v_t^{(n)}) \tag{2.10}$$

Außer dieser Aufteilung ist auch eine Zerlegung in k bzw. n getrennte Bitströme möglich, die den k Eingängen bzw. n Ausgängen im Encoder zugeordnet werden können:

$$\begin{aligned} i\text{–te Informationsfolge}: \quad & \underline{u}^{(i)} = (u_0^{(i)}, u_1^{(i)}, \ldots, u_t^{(i)}, \ldots), \quad && i = 1, \ldots, k && (2.11)\\ j\text{–te Codefolge}: \quad & \underline{v}^{(j)} = (v_0^{(j)}, v_1^{(j)}, \ldots, v_t^{(j)}, \ldots), \quad && j = 1, \ldots, n && (2.12) \end{aligned}$$

Der Zusammenhang $\underline{u}$ und $\underline{i}$ sowie $\underline{v}$ und $\underline{c}$ läßt sich wie folgt darstellen:

$$u_t^{(i)} = i_{kt+i-1} \quad i = 1, \ldots, k \tag{2.13}$$

$$v_t^{(j)} = c_{nt+j-1} \quad j = 1, \ldots, n \tag{2.14}$$

Und die Codierungsvorschrift lautet analog zu Gleichung 2.6:

$$\underline{v} = \underline{G} \cdot \underline{u}. \tag{2.15}$$

Dabei wird die Generatormatrix entsprechend der Aufteilung der Informations- und Codefolge ebenfalls in $(m+1)$ Untermatrizen aufgeteilt. Dies ist in Beispiel 2.3 bereits angedeutet:

$$\underline{G} = \begin{bmatrix} \underline{G}_0 & \underline{G}_1 & \ldots & \underline{G}_m & \underline{0} & \underline{0} & \underline{0} \ldots \\ \underline{0} & \underline{G}_0 & \underline{G}_1 & \ldots & \underline{G}_m & \underline{0} & \underline{0} \ldots \\ \underline{0} & \underline{0} & \underline{G}_0 & \underline{G}_1 & \ldots & \underline{G}_m & \underline{0} \ldots \\ \underline{0} & \underline{0} & \underline{0} & \underline{G}_0 & \underline{G}_1 & \ldots & \underline{G}_m \ldots \\ & & & \vdots & & & \ddots \end{bmatrix}. \tag{2.16}$$

Diese Teilmatrizen $\underline{G}_l$ haben die Dimension $k \times n$:

$$\underline{G}_l = \begin{bmatrix} g_l^{(1,1)} & \ldots & g_l^{(1,n)} \\ g_l^{(2,1)} & \ldots & g_l^{(2,n)} \\ \vdots & g_l^{(i,j)} & \vdots \\ g_l^{(k,1)} & \ldots & g_l^{(k,n)} \end{bmatrix}, \qquad l = 0, \ldots, m \tag{2.17}$$

Beispiel 2.4: Die Untermatrizen der Generatormatrix aus Beispiel 2.3 lauten:

$$\underline{G}_0 = \begin{bmatrix} 101 \\ 011 \end{bmatrix}, \quad \underline{G}_1 = \begin{bmatrix} 111 \\ 001 \end{bmatrix}, \quad \underline{G}_2 = \begin{bmatrix} 000 \\ 101 \end{bmatrix}$$

□

Die einzelnen Codebitvektoren $\underline{v}_t$ berechnen sich in Abhängigkeit der letzten m Informations- vektoren sowie dem aktuellen Informationsvektor:

$$\underline{v}_t = \sum_{l=0}^{m} \underline{u}_{t-l} \cdot \underline{G}_l \tag{2.18}$$

Polynomialdarstellung

In dieser Darstellungsart werden die Informationsfolge und die Codefolge als Vektoren aus k bzw. n Polynomen in D dargestellt. D ist dabei der Verzögerungsoperator und entspricht dem Verzögerungselement (Delay-Element) im Schieberegister. Man schreibt:

$$\underline{u}(D) = (u^{(1)}(D), \ldots, u^{(k)}(D)) \tag{2.19}$$

$$\underline{v}(D) = (v^{(1)}(D), \ldots, v^{(n)}(D)) \tag{2.20}$$

Die Teilbitfolgen nach 2.11 und 2.12 werden nun als (halbunendliche) Polynome dargestellt:

$$u^{(i)}(D) = u_0^{(i)} + \ldots + u_t^{(i)} D^t + \ldots, \qquad i = 1, \ldots, k \tag{2.21}$$

$$v^{(j)}(D) = v_0^{(j)} + \ldots + v_t^{(j)} D^t + \ldots, \qquad j = 1, \ldots, n \tag{2.22}$$

Die Koeffizienten $u_t^{(i)}$ und $v_t^{(j)}$ sind dabei identisch zu den Informations- bzw. Codebits in 2.13 und 2.14.

Auch in dieser Darstellung wird die Codierungsvorschrift in Form einer Matrixmultiplikation angegeben. Sie lautet:

$$\underline{v}(D) = \underline{u}(D) \cdot \underline{G}(D) \tag{2.23}$$

Die hierzu benötigte *polynomiale Generatormatrix* $G(D)$ berechnet sich aus den Untermatrizen nach Gleichung 2.17 zu

$$\underline{G}(D) = \underline{G}_0 + D\underline{G}_1 + \ldots + D^m \underline{G}_m \tag{2.24}$$

$$\underline{G(D)} = \begin{bmatrix} g^{(1,1)}(D) & g^{(1,2)}(D) & \ldots & g^{(1,n)}(D) \\ g^{(2,1)}(D) & g^{(2,2)}(D) & \ldots & g^{(2,n)}(D) \\ \vdots & \vdots & & \vdots \\ g^{(k,1)}(D) & g^{(k,2)}(D) & \ldots & g^{(k,n)}(D) \end{bmatrix}. \tag{2.25}$$

Beispiel 2.5: Für das oben betrachtete Beispiel lautet die polynomiale Generatormatrix

$$\underline{G(D)} = \begin{bmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{bmatrix}.$$

□

Die Elemente dieser Matrix sind die sogenannten *Generatorpolynome*, deren Koeffizienten auch direkt aus dem Encoder in CCF ablesbar sind (siehe Bild 2.1):

$$g^{(i,j)}(D) = g_0^{(i,j)} + D \cdot g_1^{(i,j)} + \cdots + D^{\nu_i} \cdot g_{\nu_i}^{(i,j)}; \qquad 1 \leq i \leq k, \quad 1 \leq j \leq n \tag{2.26}$$

Mit ihrer Hilfe lassen sich die einzelnen Codebitströme $v^{(j)}(D)$ folgendermaßen berechnen:

$$v^{(j)}(D) = \sum_{i=1}^{k} u^{(i)}(D) \cdot g^{(i,j)}(D), \quad j = 1, \ldots, n \tag{2.27}$$

Die Indizes i und j der Generatorpolynome geben dabei an, welche Informationsfolge $\underline{u}^{(i)}$ Einfluß auf welche Codefolgen $\underline{v}^{(j)}$ hat.

Die Berechnung der Koeffizienten $v_t^{(j)}$, also der einzelnen Codebits, kann man mit Hilfe einer diskreten Faltung darstellen (daher kommt die Bezeichnung Faltungscode):

$$v_t^{(j)} = \sum_{i=1}^{k} \underbrace{\left(\sum_{l=0}^{\nu_i} u_{t-l}^{(i)} \cdot g_l^{(i,j)} \right)}_{Faltung}, \quad j = 1, \ldots, N \tag{2.28}$$

Bei gegebenen Generatorpolynomen lassen sich die Einflußlängen ν_i wie folgt ermitteln:

$$\nu_i = \max_j \{\mathrm{grad}(g^{(i,j)}(D))\} \tag{2.29}$$

Die Generatorpolynome stehen außerdem im direkten Zusammenhang mit der oben beschriebenen Oktaldarstellung eines Faltungscodes. Der Dezimalwert der oktalen Generatorpolynome läßt sich aus den Polynomkoeffizienten $g_l^{(i,j)}$ berechnen:

$$g^{(j)} = \sum_{i=1}^{k} \sum_{l=0}^{\nu_i} g_l^{(i,j)} \cdot 2^{(m-l)k-1+i} \tag{2.30}$$

Der Codetrellis

Die Gesamteinflußlänge ν eines Faltungscodes gibt an, wieviele Speicherelemente zur Realisierung des Encoders in der Controller Canonical Form notwendig sind. Den Inhalt dieser Speicherelemente zum Zeitpunkt t bezeichnet man als den aktuellen Zustand des Encoders. Da jedes Speicherelement bei einem binären Faltungscode zwei mögliche Inhalte besitzen kann, beträgt die Anzahl möglicher Zustände

$$2^{\nu}. \tag{2.31}$$

Nach Gleichung 2.4 gilt allgemein:

$$\nu \leq m \cdot k \tag{2.32}$$

Im folgenden soll nur der Spezialfall betrachtet werden, daß alle Eingangsschieberegister die gleiche Länge besitzen, daß also alle Einflußlängen ν_i gleich groß sind. Es gilt dann

$$m = \nu_i, \qquad 1 \leq i \leq k \tag{2.33}$$

und in Gleichung 2.32 gilt das Gleichheitszeichen. Die Anzahl der möglichen Zustände beträgt für diesen Fall

$$2^{m \cdot k}. \tag{2.34}$$

Die Definition eines Zustands zum Zeitpunkt t lautet:

$$\underline{\theta}_t = (\underline{u}_{t-m}, \cdots, \underline{u}_{t-1}) \tag{2.35}$$

Die Zustände $\underline{\theta}_t$ und $\underline{\theta}_{t+1}$ zweier aufeinanderfolgender Zeitpunkte t und $t+1$ werden durch den Übergang $\underline{\gamma}_{t+1}$ verbunden, der abweichend von [Die96] wie folgt definiert wird:

$$\underline{\gamma}_{t+1} = (\underbrace{\underline{u}_{t-m}, \overbrace{\underline{u}_{t-m+1}, \ \ldots, \ \underline{u}_{t-1}}_{\underline{\theta}_t}}^{}, \overbrace{\underline{u}_t}^{\underline{\theta}_{t+1}}) \tag{2.36}$$

Jeder Übergang wird bestimmt durch seinen Startzustand $\underline{\theta}_t$ und die k letzten Informationsbits $\underline{u}_t$. Daher gehen von jedem Zustand 2^k Übergänge aus. Insgesamt existieren ausgehend von Gleichung 2.34 zwischen zwei aufeinanderfolgenden Zeitpunkten also

$$2^{m \cdot k} \cdot 2^k = 2^{(m+1) \cdot k} \tag{2.37}$$

verschiedene Übergänge. Diese Anzahl wird auch als *Zweigkomplexität* (engl.: branch complexity) bezeichnet.

Das Trellisdiagramm (der Codetrellis) stellt die graphische Darstellung der Zustände und Übergänge über der Zeit dar. Dabei werden Zustände durch Knoten und Übergänge durch Zweige symbolisiert. Bild 2.2 zeigt einen Ausschnitt aus dem Trellisdiagramm des Codes $(5, 7)$.

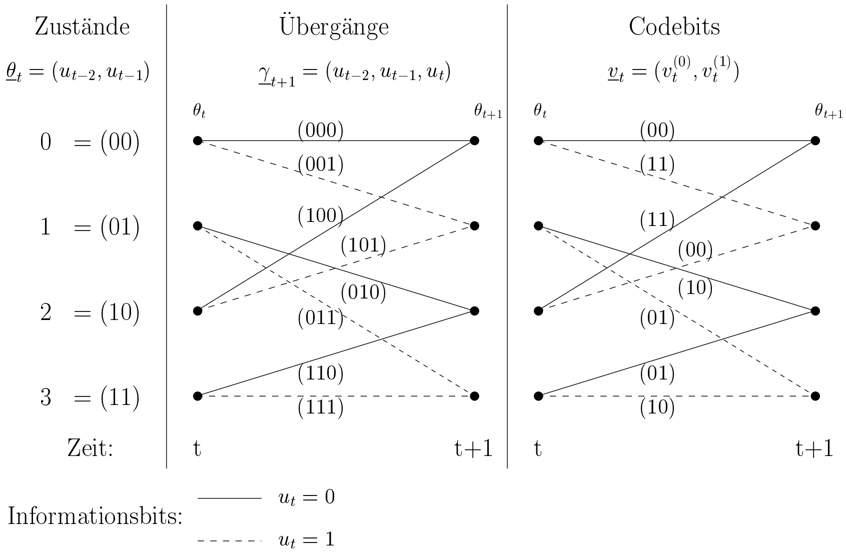


Abbildung 2.2: Ausschnitt aus dem Trellisdiagramm des Codes (5,7)

Wie bei der Binärdarstellung der Generatorpolynomkoeffizienten auf Seite 5 kann man auch die Vektoren $\underline{\theta}_t$ und $\underline{\gamma}_t$ als Binärzahl auffassen. Die entsprechende Dezimalzahl dient zur Numerierung der Zustände und Übergänge und soll im folgenden gleichwertig zur Schreibweise als binärer Zeilenvektor verwendet werden.
Sowohl Zustände als auch Übergänge sind nur von der Informationsfolge abhängig. Die Zuordnung der Informationsbits zu den Trellisübergängen wird durch folgende Gleichung beschrieben, die man aus 2.36 ableiten kann:

$$\underline{u}_t = \underline{\gamma}_{t+1} \cdot \begin{pmatrix} \underline{0}_{[k]} \\ \vdots \\ \underline{0}_{[k]} \\ \underline{I}_{[k]} \end{pmatrix} = (\underline{u}_{t-m}, \underline{u}_{t-m+1}, \dots, \underline{u}_{t-1}, \underline{u}_t) \cdot \begin{pmatrix} \underline{0}_{[k]} \\ \vdots \\ \underline{0}_{[k]} \\ \underline{I}_{[k]} \end{pmatrix} \tag{2.38}$$

Dabei bezeichnet $\underline{I}_{[k]}$ die $(k \times k)$-Einheitsmatrix und $\underline{0}_{[k]}$ die $(k \times k)$-Nullmatrix. Nach dieser Zuordnung sind die Informationsbits direkt aus den letzten k Stellen (Bits) eines Übergangs ablesbar. Im Beispiel in Bild 2.2 ist wegen $k = 1$ jedem Übergang genau ein Informationsbit u_t zugeordnet. Dessen Wert ist daran zu erkennen, ob der Übergang gestrichelt oder durchgezogen gezeichnet ist.
Außerdem sind jedem Übergang n Codebits zugeordnet (siehe auch Gleichung 2.18):

$$\underline{v}_t = \underline{\gamma}_{t+1} \cdot \begin{pmatrix} \underline{G}_m \\ \vdots \\ \underline{G}_0 \end{pmatrix} = (\underline{u}_{t-m}, \underline{u}_{t-m+1}, \dots, \underline{u}_{t-1}, \underline{u}_t) \cdot \begin{pmatrix} \underline{G}_m \\ \vdots \\ \underline{G}_0 \end{pmatrix} \tag{2.39}$$

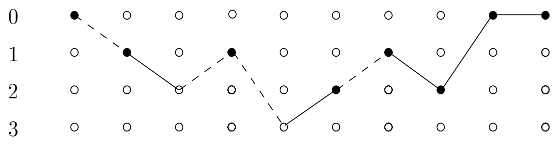
Ein sogenannter *Pfad durch das Trellisdiagramm* ist fest verbunden mit einer bestimmten Folge von Informationsbits sowie einer Codebitfolge und kann auf mehrere Arten dargestellt werden. Eine Möglichkeit ist, die der Reihe nach durchlaufenen Zustände anzugeben (siehe Bild 2.3):

$$\underline{\theta} = (\theta_{t_0}, \theta_{t_0+1}, \dots, \theta_t, \dots) \tag{2.40}$$

Gleichwertig dazu ist die Angabe der nacheinander durchlaufenen Übergänge:

$$\underline{\gamma} = (\gamma_{t_0}, \gamma_{t_0+1}, \dots, \gamma_t, \dots) \tag{2.41}$$

Ist nichts anderes vermerkt, so soll im folgenden davon ausgegangen werden, daß die beschriebenen Pfade im Nullzustand beginnen, und der erste angegebene Zustand $\underline{\theta}_{t_0}$ oder Übergang $\underline{\gamma}_{t_0}$ eine Abweichung aus dem Nullzustand einleitet.



$\underline{\theta} = ($ 1, 2, 1, 3, 2, 1, 2, 0, 0, $\dots)$

$\underline{\gamma} = ($ 1, 2, 5, 3, 6, 5, 2, 4, 0, $\dots)$

Abbildung 2.3: Beschreibung eines Pfades im Trellisdiagramm des Codes (5,7)

2.1.2 Spezielle Schreibweisen für Sonderfälle

In der vorliegenden Arbeit werden in erster Linie Codes der Rate $1/n$ betrachtet. Diese werden häufig als Unit Memory Codes dargestellt. Aus diesem Grund sollen hier nun die wichtigsten Vereinfachungen und spezielle Schreibweisen beschrieben werden, die für diese Fälle gelten.

Codes der Rate $1/n$

Codes mit der Dimension $k = 1$ werden ab sofort auch als *konventionelle Codes* bezeichnet. Für sie ergeben sich folgende Vereinfachungen im Vergleich zum allgemeinen Fall:

- Es existiert nur ein Eingangsschieberegister in der Controller Canonical Form und deshalb gilt:

$$\nu = \nu_1 = m \tag{2.42}$$

- Die Vektoren $\underline{u}_t$ der Informationsfolge bestehen für $k = 1$ aus nur einem Element. Bei diesem wird daher zur Vereinfachung der obere Index (1) weggelassen, also statt $u_t^{(1)}$ nur u_t geschrieben.

$$\underline{u} = (u_0, u_1, u_2, \ldots, u_t, \ldots), \qquad u_t = 0 \text{ für } t < 0 \tag{2.43}$$

  Gleichung 2.13 vereinfacht sich zu:

$$u_t = i_t \tag{2.44}$$

  Die Teilmatrizen $\underline{G}_l$ besitzen jeweils nur noch eine Zeile.

- Das Trellisdiagramm von Codes mit $k = 1$ soll als *konventioneller Trellis* bezeichnet werden. In ihm existieren $2^m = 2^\nu$ Zustände, und die Anzahl der Übergänge zu einem Zeitpunkt beträgt $2^{m+1} = 2^{\nu+1}$, ist also doppelt so groß wie die Anzahl der Zustände.

  Jedem Übergang ist genau ein Informationsbit zugeordnet, Gleichung 2.38 vereinfacht sich daher wie folgt:

$$u_t = \underline{\gamma}_{t+1} \cdot \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} = (u_{t-m}, u_{t-m+1}, \ldots, u_{t-1}, u_t) \cdot \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{2.45}$$

Unit Memory Codes

Jeder Faltungscode $\mathcal{C}(n, k, d)$ der Rate k/n, freier Distanz d und Gedächtnislänge m läßt sich außer über die bisher beschriebenen Methoden auch als *Unit Memory Code* (UM-Code), das heißt als Code mit Gedächtnislänge $M = 1$, darstellen (siehe z.B. [Lee76],[Mau94]).

Man erhält eine UM-Darstellung $\mathcal{C}_{\mathrm{UM}}(N = K \cdot n, K, d)$ eines konventionellen Codes $\mathcal{C}$, wenn man die Dimension $K \geq m$ wählt.

Zur Unterscheidung von der herkömmlichen Schreibweise werden für die UM-Darstellung in dieser Arbeit andere Bezeichnungen verwendet, für manche Parameter werden einfach die

	herkömmlich	UM-Darstellung
Einflußlänge	ν_i	$\mathcal{V}_i$
Gesamteinflußlänge	ν	$\mathcal{V}$
Gedächtnislänge	m	M
Dimension	k	K
	n	N
Informationsfolge	$\underline{u}$	$\underline{x}$
Codefolge	$\underline{v}$	$\underline{y}$
Zeit	t	τ
Zustand im Trellis	$\underline{\theta_t}$	$\underline{\Theta_\tau}$
Übergang im Trellis	$\underline{\gamma_t}$	$\underline{\Gamma_\tau}$

Tabelle 2.1: Schreibweise für Unit Memory Codes

entsprechenden Großbuchstaben verwendet. In Tabelle 2.1 sind die wichtigsten Unterschiede der beiden Darstellungsarten zusammengefaßt.

Neben den Unterschieden in der Darstellung ergeben sich auch bestimmte Vereinfachungen:

- Bei UM-Codes handelt es sich definitionsgemäß um Faltungscodes mit Gedächtnislänge $M = 1$. Aus den Gleichungen 2.4 und 2.3 ergibt sich:

$$M = \mathcal{V}_i = 1, \qquad i = 1 \dots K \tag{2.46}$$
$$\mathcal{V} = K \tag{2.47}$$

- Für die Informations- und Codefolge ergibt sich eine neue Aufteilung in Vektoren der Größe K bzw N:

$$\underline{x} = (\underline{x}_0, \underline{x}_1, \underline{x}_2, \dots, \underline{x}_\tau, \dots), \qquad \underline{x}_\tau = \underline{0} \text{ für } \tau < 0 \tag{2.48}$$
$$\underline{y} = (\underline{y}_0, \underline{y}_1, \underline{y}_2, \dots, \underline{y}_\tau, \dots), \qquad \underline{y}_\tau = \underline{0} \text{ für } \tau < 0 \tag{2.49}$$

$$\underline{x}_\tau = (x_\tau^{(1)}, \dots, x_\tau^{(K)}) \tag{2.50}$$
$$\underline{y}_\tau = (y_\tau^{(1)}, \dots, y_\tau^{(N)}) \tag{2.51}$$

$$\underline{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, \dots, x_\tau^{(i)}, \dots), \quad i = 1, \dots, K \tag{2.52}$$
$$\underline{y}^{(j)} = (y_0^{(j)}, y_1^{(j)}, \dots, y_\tau^{(j)}, \dots), \quad j = 1, \dots, N \tag{2.53}$$

Die Codierungsvorschrift in UM-Schreibweise lautet :

$$\underline{y} = \underline{G} \cdot \underline{x} \tag{2.54}$$

Die halbunendliche Generatormatrix ist für die UM-Darstellung identisch zu der in herkömmlicher Schreibweise. Allerdings wird sie in andere Untermatrizen aufgeteilt. Es existieren nur noch zwei Untermatrizen $\underline{G}_0$ und $\underline{G}_1$ mit den Dimensionen $K \times N$:

$$\underline{G} = \begin{bmatrix} \underline{G}_0 & \underline{G}_1 & \underline{0} & \dots & \underline{0} & \dots \\ \underline{0} & \underline{G}_0 & \underline{G}_1 & \underline{0} & \underline{0} & \dots \\ \underline{0} & \underline{0} & \underline{G}_0 & \underline{G}_1 & \underline{0} & \dots \\ & & \vdots & & & \ddots \end{bmatrix} \tag{2.55}$$

Beispiel 2.6:

Für den Code aus Beispiel 2.3 und Bild 2.1 erhält man für $K = \nu = 4$ die Untermatrizen:

$$\underline{G}_0 = \begin{bmatrix} 101110 \\ 011001 \\ 000101 \\ 000011 \end{bmatrix}, \quad \underline{G}_1 = \begin{bmatrix} 000000 \\ 101000 \\ 110000 \\ 001101 \end{bmatrix}$$

□

- Die Umrechnung nach Gleichung 2.13 und 2.14 gilt prinzipiell auch hier, die Umrechnung von $\underline{u}$ nach $\underline{x}$ und von $\underline{v}$ nach $\underline{y}$ ist etwas komplizierter:

$$x_\tau^{(l)} = i_{K\tau+l-1} = u_{K\tau+l-1 \text{ div } k}^{([K\tau+l-1 \bmod k]+1)} \qquad l = 1, \dots, K \tag{2.56}$$

$$y_\tau^{(j)} = c_{N\tau+j-1} = v_{N\tau+j-1 \text{ div } n}^{([N\tau+j-1 \bmod n]+1)} \qquad j = 1, \dots, N \tag{2.57}$$

Für den wichtigen Spezialfall $k = 1$ vereinfacht sich die Umrechnung:

$$x_\tau^{(l)} = u_{K\tau+l-1} \qquad l = 1, \dots, K \tag{2.58}$$

- Auch die Polynomschreibweise ändert sich für UM-Codes entsprechend Tabelle 2.1

$$\underline{x}(D) = (x^{(1)}(D), \dots, x^{(K)}(D)) \tag{2.59}$$
$$\underline{y}(D) = (y^{(1)}(D), \dots, y^{(N)}(D)) \tag{2.60}$$

$$x^{(i)}(D) = x_0^{(i)} + \dots + u_\tau^{(i)} D^\tau + \dots, \quad i = 1, \dots, K \tag{2.61}$$
$$v^{(j)}(D) = y_0^{(j)} + \dots + y_\tau^{(j)} D^\tau + \dots, \quad j = 1, \dots, N \tag{2.62}$$

Und die Codierungsvorschrift lautet:

$$\underline{y}(D) = \underline{x}(D) \cdot \underline{G}(D) \tag{2.63}$$

Die polynomiale Generatormatrix setzt sich aus den Matrizen $\underline{G}_0$ und $\underline{G}_1$ zusammen:

$$\underline{G}(D) = \underline{G}_0 + D\underline{G}_1 \tag{2.64}$$

Beispiel 2.7: Im obigen Beispiel erhält man

$$\underline{G}(D) = \begin{bmatrix} 1 & & 1 & 1 & 1 & 0 \\ D & 1 & 1+D & 0 & 0 & 1 \\ D & D & 0 & 1 & 0 & 1 \\ 0 & 0 & D & D & 1 & 1+D \end{bmatrix}$$

□

- Der Trellis eines UM-Codes, den man auch als *UM-Trellis* bezeichnen kann, unterscheidet sich im allgemeinen vom Trellis der konventionellen Darstellung. Für einen UM-Code existieren im Trellisdiagramm $2^{\mathcal{V}} = 2^K$ Zustände und die Zahl der Übergänge zwischen zwei Zeitpunkten beträgt $2^{(M+1)\cdot K} = 2^{2\cdot K}$.

  Nach der Definition eines Trellisübergangs in 2.36 ergibt sich für die Übergänge im UM-Trellis:

$$\underline{\Gamma}_{\tau+1} = \left(\underline{x}_{\tau-1}, \underline{x}_{\tau}\right) = \left(\underline{\Theta}_{\tau}, \underline{\Theta}_{\tau+1}\right) \tag{2.65}$$

  Und die Zuordnung der Informationsbits zu den Trellisübergängen nach Gleichung 2.38 vereinfacht sich wie folgt:

$$\underline{x}_{\tau} = \underline{\Gamma}_{\tau+1} \cdot \begin{pmatrix} \underline{0}_{[K]} \\ \underline{I}_{[K]} \end{pmatrix} \tag{2.66}$$

  Wie man aus 2.66 und 2.65 ablesen kann, gilt für UM-Codes:

$$\underline{x}_{\tau} = \underline{\Theta}_{\tau+1} \tag{2.67}$$

  Das bedeutet, daß der Zustand des Encoders zum Zeitpunkt $\tau + 1$ ausschließlich vom letzten Informationsvektor $\underline{x}_{\tau}$ abhängt. Da dieser frei wählbar ist, kann innerhalb eines Übergangs jeder beliebige Zustand im UM-Trellis erreicht werden.

2.1.3 Rekursive Faltungscodes

Rekursive Faltungscodes[2] sollen in dieser Arbeit nur am Rande betrachtet werden, daher erfolgen hier nur wenige grundlegende Erläuterungen.

Bei rekursiven Faltungscodes existieren außer den Generatorpolynomen zusätzlich sogenannte Rückkoppelpolynome $q^{(i)}(D)$, welche die Registerinhalte der Schieberegister logisch verknüpfen und an ihren Eingang zurückkoppeln (siehe Bild 2.4).

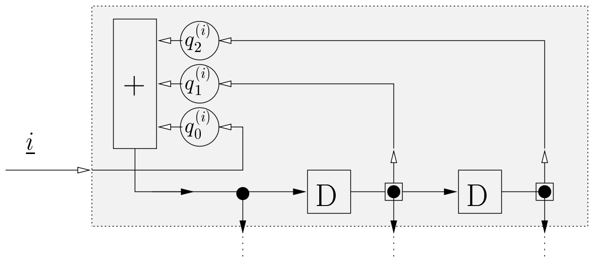


Abbildung 2.4: Rückgekoppeltes Schieberegister eines rekursiven Faltungscodes mit $\nu_i = 2$

Da in der Controller Canonical Form eines Codes k Eingangsschieberegister existieren, gibt es auch maximal k verschiedene Rückkoppelpolynome. Die Generatormatrix eines rekursiven Faltungscodes in Polynomschreibweise kann daher immer auf die folgende Form

[2] siehe auch [Bos94], IIR-Systeme, Rekursives System

gebracht werden, in der k verschiedene Zähler existieren. Diese entsprechen den Rückkoppelpolynomen:

$$\underline{G(D)} = \begin{bmatrix} \frac{g^{(1,1)}(D)}{q^{(1)}(D)} & \frac{g^{(1,2)}(D)}{q^{(1)}(D)} & \cdots & \frac{g^{(1,n)}(D)}{q^{(1)}(D)} \\ \vdots & \vdots & & \vdots \\ \frac{g^{(k,1)}(D)}{q^{(k)}(D)} & \frac{g^{(k,2)}(D)}{q^{(k)}(D)} & \cdots & \frac{g^{(k,n)}(D)}{q^{(k)}(D)} \end{bmatrix} \tag{2.68}$$

In dieser Arbeit interessiert in erster Linie der Fall, daß alle Eingangsschieberegister das gleiche Rückkoppelpolynom besitzen, also $q^{(1)}(D) = \ldots = q^{(k)}(D)$ gilt.

Beispiel 2.8: Ersetzt man die beiden Schieberegister in Bild 2.1 durch das rückgekoppelte Schieberegister aus Bild 2.4 und wählt $q^{(i)}(D) = 1 + D^3$, so erhält man die Realisierung des rekursiven Faltungscodes mit der Generatormatrix

$$\underline{G(D)} = \begin{bmatrix} \frac{1+D}{1+D^3} & \frac{D}{1+D^3} & \frac{1}{1+D^3} \\ \frac{D^2}{1+D^3} & \frac{1}{1+D^3} & \frac{1+D+D^2}{1+D^3} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+D+D^2} & \frac{D}{1+D^3} & \frac{1}{1+D^3} \\ \frac{D^2}{1+D^3} & \frac{1}{1+D^3} & \frac{1}{1+D} \end{bmatrix}.$$

□

Nichtrekursive Faltungscodes sind ein Sonderfall von rekursiven Faltungscodes, für die alle Eingangsschieberegister das einfache Rückkoppelpolynom $q^{(i)}(D) = 1$ besitzen.

2.1.4 Punktierte Faltungscodes

Um Faltungscodes hoher Raten zu erhalten, die mit wenig Aufwand, also geringer Komplexität decodiert werden können, werden oft niederratige Faltungscodes punktiert (siehe [Hol88], [WHPH87] und [YKH84]). Dieses Punktieren entspricht dem Löschen einzelner Bits aus der Codefolge, die der Encoder des niederratigen Faltungscodes liefert.

Das Muster, nach dem Codebits entfernt werden, wird in Form der sogenannten Punktierungsmatrix angegeben. Es handelt sich dabei um eine binäre Matrix mit n Zeilen und p_{pkt} Spalten, in der z_{pkt} Elemente gleich 1 sind. p_{pkt} wird im weiteren als Punktierungsperiode bezeichnet.

Die Punktierungsmatrix wird mit der Zeit t spaltenweise durchlaufen. Steht dabei in der Punktierungsmatrix in Zeile j der aktuellen Spalte eine 1, so wird das Codebit $v_t^{(j)}$ des aktuellen Codebitvektors übertragen, bei einer 0 nicht. Die j-te Zeile bezieht sich also auf den j-ten Ausgang des Schieberegisters. Nach Erreichen der letzten Spalte, also nach p_{pkt} Codevektoren bzw. $p_{\text{pkt}} \cdot n$ Codebits, wird wieder in der ersten Spalte begonnen.

Beispiel 2.9: Um mit dem Code $(5, 7)$ der Rate $1/2$ durch Punktierung einen Code der Rate $4/5$ zu konstruieren, kann man beispielsweise folgende Punktierungsmatrix aus [YKH84] verwenden.

$$\underline{P} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}; \qquad p_{\text{pkt}} = 4, \quad z_{\text{pkt}} = 5$$

Diese Matrix hat folgende Punktierung zur Folge:

$$\begin{aligned} \text{Codefolge:} \quad & \underline{c} = (v_0^{(1)}, v_0^{(2)}, v_1^{(1)}, v_1^{(2)}, v_2^{(1)}, v_2^{(2)}, v_3^{(1)}, v_3^{(2)}, v_4^{(1)}, v_4^{(2)}, v_5^{(1)} \dots) \\ \text{punktierte Folge:} \quad & \underline{c}_{\text{pkt}} = (v_0^{(1)}, v_0^{(2)}, \quad v_1^{(2)}, v_2^{(1)}, \quad , v_3^{(1)}, \quad , v_4^{(1)}, v_4^{(2)}, \quad \dots) \end{aligned}$$

Da von je 8 Codebits nur 5 übertragen werden, erhöht sich die Coderate auf

$$r_{\text{pkt}} = \frac{1}{2} \cdot \frac{2 \cdot 4}{5} = \frac{4}{5}.$$

□

Im allgemeinen ist das Resultat der Punktierung eines Codes der Rate $r = k/n$ mit der Punktierungsperiode p_{pkt} ein Code der Rate

$$r_{\text{pkt}} = r \cdot \frac{n \cdot p_{\text{pkt}}}{z_{\text{pkt}}} = k \cdot \frac{p_{\text{pkt}}}{z_{\text{pkt}}}. \tag{2.69}$$

Punktierte Codes können mit demselben Decoder wie ihr Muttercode decodiert werden. Im Empfänger muß dazu natürlich das Punktierungsmuster, das heißt die Punktierungsmatrix, bekannt sein. Statt der nicht übertragenen Bits werden dann je nach Decoder geeignete Werte in die Folge der Empfangswerte eingefügt (siehe auch Kapitel 4.2).

Zur Vereinfachung soll für punktierte Codes die folgende, kurze Schreibweise gelten:

$$(g^{(1)}, g^{(2)}, \dots, g^{(n)}; \quad r_{\text{pkt}}) \tag{2.70}$$

Das heißt, neben der Oktaldarstellung des Muttercodes wird nur die Coderate des punktierten Codes angegeben, wenn eindeutig ist, welche Punktierungsmatrix verwendet wird.

2.1.5 Terminierte Faltungscodes

Im Gegensatz zu den Codeworten eines Blockcodes, die alle eine gleiche, endliche Länge besitzen und getrennt voneinander decodiert werden (siehe z.B. [Bos92]), ist die Codefolge eines Faltungscodes ebenso wie die Informationsfolge prinzipiell unendlich lang (siehe 2.11 und 2.12). Bestimmte Decodierverfahren, insbesondere die MAP-Decodierung, wie sie in [BCJR74] beschrieben wird, können jedoch nur für die Decodierung einer endlichen Codefolge eingesetzt werden (siehe auch Kapitel 4.2). Aus diesem Grund wird oft eine künstliche *Terminierung* von Faltungscodes vorgenommen. Darunter versteht man das Überführen des Encoders in einen definierten Zustand, im allgemeinen in den Nullzustand. Dies entspricht dem Rücksetzen aller Speicherinhalte und kann nach Gleichung 2.35 durch das Einfügen von

$$l_{\text{Term}} = m \cdot k \tag{2.71}$$

Nullen also von m Vektoren $\underline{u}_t = \underline{0}$ in die Informationsfolge erreicht werden. Geschieht dies jeweils periodisch nach einer festen Zahl von Informationsbits, so erhält man vergleichbar zu Blockcodes aufeinanderfolgende, unabhängige Blöcke von Codebits. Die Länge dieser Blöcke ist jedoch meist wesentlich größer als bei Blockcodes.

Einen einzelnen solchen Block, der zum Zeitpunkt $t = 0$ beginnt, kann man als Ausschnitt aus der (halb-)unendlichen Informations- und Codefolge darstellen:

$$\text{Informationsfolge:} \quad \hat{\underline{u}}_0 = (\underline{u}_0, \underline{u}_1, \underline{u}_2, \dots, \underline{u}_{L-1}) \tag{2.72}$$
$$\text{Codefolge:} \quad \hat{\underline{v}}_0 = (\underline{v}_0, \underline{v}_1, \underline{v}_2, \dots, \underline{v}_{L-1}) \tag{2.73}$$

Die *Terminierungsbits* sind Teil des Informationsblocks, d.h. es gilt:

$$\underline{u}_{L-m} = \ldots = \underline{u}_{L-1} = \underline{0} \tag{2.74}$$

Wählt man die Bits in allen anderen Informationsblöcken $\underline{\hat{u}}_i$ zu Null, also

$$\underline{u}_t = \underline{0} \quad \text{für } t < 0 \text{ oder } t \geq L,$$

so werden auch alle zugehörigen Codesequenzen $\underline{\hat{v}}_i$ zu Nullfolgen:

$$\underline{v}_t = \underline{0} \quad \text{für } t < 0 \text{ oder } t \geq L$$

In dieser Beschreibung ist L die *Blocklänge des terminierten Faltungscodes in Trellisübergängen* oder Informations- bzw. Codevektoren.

Die *Blocklänge in Informationsbits* (einschließlich der Terminierungsbits l_{Term}) lautet

$$l_{\mathrm{Info}+} = k \cdot L \tag{2.75}$$

$$= l_{\mathrm{Info}} + l_{\mathrm{Term}} \tag{2.76}$$

und die *Blocklänge in Codebits*

$$l_{\mathrm{Code}} = n \cdot L. \tag{2.77}$$

KAPITEL

3

Partitionierung von Faltungscodes

Die Idee der verallgemeinerten Codeverkettung ist der Schutz der Numerierung eines inneren Teilcodes durch einen äußeren Code. Die Grundlage zur verallgemeinerten Codeverkettung ist daher die Partitionierung, die Aufteilung eines inneren Codes in mehrere Untercodes und deren Numerierung.
In [Bos92] sind verschiedene Methoden zur Partitionierung von Blockcodes in Untercodes mit maximalen Mindestdistanzen angegeben. In diesem Kapitel wird nach einigen grundlegenden Definitionen eine Methode zur Partitionierung von Faltungscodes mit Hilfe der Informationsfolge beschrieben, wie sie prinzipiell in [BDS96a] und [BDS96b] verwendet wird. Dabei wird zunächst die grundlegende Idee zur Partitionierung eines terminierten Faltungscodes erläutert. Da dieses Verfahren zunächst nur eine zufällige Aufteilung in Untercodes darstellt, wird im nächsten Schritt die Methode der gezielten „Pfad-Punktierung“ vorgestellt, die eine Maximierung der freien Distanzen der Untercodes ermöglicht.
Zur Vereinfachung der Darstellung werden zunächst nur Faltungscodes der Rate $1/n$ betrachtet. Als höherratige innere Codes werden später noch punktierte Codes betrachtet.

3.1 Grundlegende Definitionen zur Partitionierung

Unter der *Partitionierung* eines Codes $\mathcal{B}^{(1)}$ versteht man die Aufteilung der Menge aller möglichen Codeworte oder -sequenzen dieses Codes in s_1 disjunkte Untermengen $\mathcal{B}^{(2)}_{\underline{z}^{(1)}}$, für die gilt:

$$\mathcal{B}^{(1)} = \bigcup_{\underline{z}^{(1)}=0}^{s_1-1} \mathcal{B}^{(2)}_{\underline{z}^{(1)}} \tag{3.1}$$

Jede der Untermengen $\mathcal{B}^{(2)}_{\underline{z}^{(1)}} \subset \mathcal{B}^{(1)}$ stellt einen sogenannten *Untercode* dar. Der Index $\underline{z}^{(1)}$ dient der Numerierung der Untercodes und kann die Werte $0, \ldots, s_1 - 1$ annehmen (siehe Gleichung 3.1). Der Unterstrich in der Schreibweise des Index $\underline{z}^{(1)}$ soll darauf hinweisen, daß die Numerierung der Untercodes im folgenden nicht nur durch Dezimalzahlen, sondern in erster Linie mit Hilfe der zugehörigen Binärzahlen bzw. binären Vektoren erfolgen soll.

Besitzt jeder der Untercodes $\mathcal{B}^{(2)}_{\underline{z}^{(1)}}$ die Mächtigkeit eins, besteht also aus nur einem Codewort, so handelt es sich um den einfachsten Fall einer Partitionierung, die Partitionierung des Codes in seine Codeworte oder -sequenzen $\underline{y}_{\underline{z}^{(1)}}$. Man spricht dann von einer *Partitionierung erster Ordnung.*

Bestehen die Untercodes $\mathcal{B}^{(2)}_{\underline{z}^{(1)}}$ dagegen aus mehr als je einem Codewort, so kann man jeden dieser Untercodes wiederum in s_2 Untercodes $\mathcal{B}^{(3)}_{\underline{z}^{(1)},\underline{z}^{(2)}}$ aufteilen, welche man auch wieder partitionieren kann, usw. Man nennt dies *mehrfache Partitionierung* von $\mathcal{B}^{(1)}$.

Allgemein bezeichnet man die Anzahl der Nummern oder Vektoren $\underline{z}^{(i)}$, die man bei mehrfacher Partitionierung zur Numerierung einer beliebigen Codesequenz $\underline{y} \in \mathcal{B}^{(1)}$ benötigt, als *Ordnung der Partitionierung.*

Für eine Partitionierung K-ter Ordnung oder K-fache Partitionierung gilt:

$$\underline{y}_{\underline{z}^{(1)},\ldots,\underline{z}^{(K)}} \in \mathcal{B}^{(K)}_{\underline{z}^{(1)},\ldots,\underline{z}^{(K-1)}} \subset \ldots \subset \mathcal{B}^{(i)}_{\underline{z}^{(1)},\ldots,\underline{z}^{(i-1)}} \subset \ldots \subset \mathcal{B}^{(1)} \tag{3.2}$$

Als i-te Partitionierungsstufe soll dabei die Partitionierung von $\mathcal{B}^{(i)}_{\underline{z}^{(1)},\ldots,\underline{z}^{(i-1)}}$ in s_i Untercodes mit Hilfe von $\underline{z}^{(i)}$ bezeichnet werden. Für sie gilt:

$$\mathcal{B}^{(i)}_{\underline{z}^{(1)},\ldots,\underline{z}^{(i-1)}} = \bigcup_{\underline{z}^{(i)}=0}^{s_i-1} \mathcal{B}^{(i+1)}_{\underline{z}^{(1)},\ldots,\underline{z}^{(i-1)},\underline{z}^{(i)}} \tag{3.3}$$

In Bild 3.1 auf Seite 21 ist zur Veranschaulichung eine 2-fache Partitionierung in je 8 Untercodes ($s_1 = s_2 = 8$) dargestellt.

Aufgrund der Tatsache, daß die Untercodes einer Partitionierungsstufe disjunkte Mengen darstellen und daß es sich bei $\mathcal{B}^{(1)}$ um einen linearen Code handelt, wird klar, daß in jeder Stufe genau ein Untercode existiert, der die Nullsequenz enthält. Nur dieser Untercode, der im folgenden $\mathcal{B}^{(i)}$ genannt wird, ist linear. Alle anderen Untercodes sind nichtlinear.

Bezeichnet man mit $d_z^{(i)}$ die freie Distanz eines Untercodes $\mathcal{B}_z^{(i)}$, so kann man die *freie Distanz der i-ten Partitionierungsstufe* definieren als die kleinste freie Distanz aller ihrer Ausgangscodes $\mathcal{B}_z^{(i)}$:

$$d^{(i)} = \min_z \left\{ d_z^{(i)} \right\} \tag{3.4}$$

Für die freien Distanzen der einzelnen Partitionierungsstufen gilt:

$$d^{(K)} \geq \ldots \geq d^{(i)} \geq \ldots \geq d^{(1)} = d \tag{3.5}$$

Für die Konstruktion von verallgemeinert verketteten Codes ist es wichtig, die Partitionierung des inneren Codes $\mathcal{B}^{(1)}$ so zu wählen, daß die freie Distanz der jeweils nächsten Partitionierungsstufe maximal ist. Das heißt, der Anstieg der freien Distanz nach Gleichung 3.5 soll von Stufe zu Stufe möglichst groß sein.

Erzielt man im $i-$ten Partitionierungsschritt die maximal erreichbare freie Distanz $d^{(i)}$, so spricht man von *optimaler Partitionierung* (vgl. [Bos92]). Wie diese optimale Partitionierung erzielt werden kann, ist eines der Hauptthemen der vorliegenden Arbeit und wird in den folgenden Abschnitten beschrieben.
Dazu wird als erstes die Grundidee zur Partitionierung von Faltungscodes nach [BDS96a] und [BDS96b] vorgestellt.

3.2 Zufällige Partitionierung von Faltungscodes

Um die grundlegende Idee bei der Partitionierung eines Faltungscodes nach [BDS96a] zu verdeutlichen, wird zunächst die Partitionierung eines *terminierten* Faltungscodes betrachtet. Dieser Spezialfall kann als Partitionierung eines Blockcodes betrachtet werden und ermöglicht daher eine anschauliche Darstellung. Desweiteren hat er große praktische Bedeutung, da die Decodierung der Untercodes in der vorliegenden Arbeit durch MAP-Decoder erfolgt, wozu eine Terminierung der Faltungscodes erforderlich ist (siehe Kapitel 4.2).

Um eine K-fache Partitionierung für einen terminierten Faltungscode $\mathcal{B}^{(1)}$ der Rate $1/n$ durchzuführen, stellt man diesen zunächst als Unit Memory Code der Dimension K dar (siehe Abschnitt 2.1.2). Wählt man die Blocklänge in Informationsbits als Vielfaches von K, also $l_{\text{Info}} = K * L$, so kann man die Informationssequenz für einen Block nach Gleichung 2.72 wie folgt angeben:

$$\underline{x} = (\underline{x}_0, \underline{x}_1, \underline{x}_2, \ldots, \underline{x}_{L-1}) \tag{3.6}$$

Diese läßt sich nach Gleichung 2.52 in K unabhängige Sequenzen zerlegen:

$$\underline{x}^{(j)} = (x_0^{(j)}, x_1^{(j)}, \ldots, x_{L-1}^{(j)}), \qquad j = 1, \ldots, K \tag{3.7}$$

Diese K Sequenzen kann man nun zur Numerierung der Untercodes verwenden, das heißt man kann

$$\underline{z}^{(i)} = \underline{x}^{(i)} \tag{3.8}$$

wählen und erhält so eine K-fache Partitionierung mit Hilfe der Informationssequenz. Da jede Folge $\underline{x}^{(i)}$ aus L Bit besteht, wird jeder Code $\mathcal{B}^{(i)}_{\ldots}$ in

$$s_1 = \ldots = s_K = s = 2^L \tag{3.9}$$

Untercodes partitioniert. Die K-te Partitionierungsstufe stellt schließlich die Numerierung der insgesamt $2^{K \cdot L}$ verschiedenen Codesequenzen $\underline{y}$ dar.

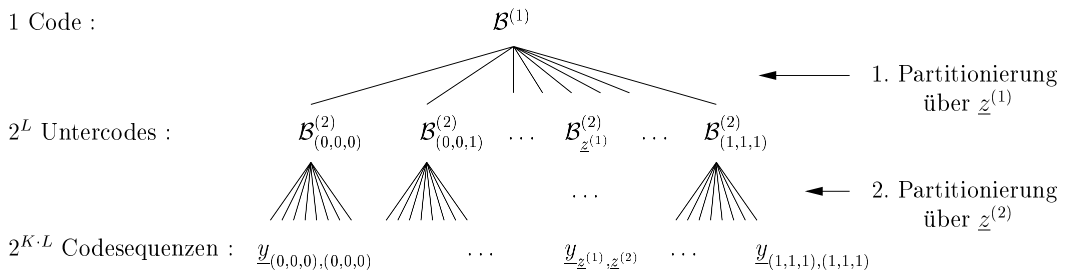


Abbildung 3.1: 2-fache Partitionierung eines term. Faltungscodes mit Blocklänge $L = 3$

Bild 3.1 zeigt als Beispiel die 2-fache Partitionierung eines terminierten Faltungscodes. Die Blocklänge wurde dabei extrem kurz gewählt ($L = 3$), um die Anzahl der Untercodes zu begrenzen. In der Praxis werden wesentlich größere Blocklängen verwendet, was eine Aufteilung eines Codes in wesentlich mehr Untercodes (2^L) zur Folge hat.
Bei Verwendung eines geeigneten Decodierverfahrens kann man auch nichtterminierte Faltungscodes partitionieren, wie oben beschrieben. Das heißt das Verfahren funktioniert auch für $L \longrightarrow \infty$. Lediglich die Darstellung als (endliches) Baumdiagramm nach Bild 3.1

ist dann so nicht mehr möglich, da auch die Anzahl 2^L der Untercodes unendlich groß wird.

Faßt man die Elemente $z_t^{(i)}$ der Numerierungsfolgen $\underline{z}^{(i)}$ analog zu denen der Informationsfolge $\underline{x}$ zusammen zu

$$\underline{z} = (\underline{z}_0, \underline{z}_1, \dots, \underline{z}_\tau, \dots) \qquad \text{mit} \quad \underline{z}_\tau = (z_t^{(1)}, \dots, z_t^{(K)}), \tag{3.10}$$

so kann man statt Gleichung 3.8 vereinfachend auch schreiben:

$$\underline{z} = \underline{x} \tag{3.11}$$

Anschaulich bedeutet die Partitionierung nach Gleichung 3.8 bzw. 3.11 folgendes:
Ein Untercode $\mathcal{B}^{(i)}_{\underline{z}^{(0)},\dots,\underline{z}^{(i-1)}}$ der $(i-1)$-ten Partitionierungsstufe ist die Menge aller Codesequenzen $\underline{y}$, für deren Informationsfolge $\underline{x}$ gilt:

- Die ersten $i-1$ Informations-Teilfolgen $\underline{x}^{(1)}, \dots, \underline{x}^{(i-1)}$ sind identisch zu den Numerierungen $\underline{z}^{(1)}, \dots, \underline{z}^{(i-1)}$ des Untercodes.

- Die Bits der restlichen Informationsfolgen $\underline{x}^{(i)}, \dots, \underline{x}^{(K)}$ können beliebige Werte (0,1) annehmen.

Die so beschriebenen Untercodes $\mathcal{B}^{(i)}_{\underline{z}^{(1)},\dots,\underline{z}^{(i-1)}}$ besitzen die folgenden Parameter:

$$\text{Dimension :} \qquad K^{(i)} = K - i + 1 \tag{3.12}$$

$$\text{Coderate :} \qquad R^{(i)} = (K - i + 1)/N \tag{3.13}$$

Wie in Abschnitt 3.1 beschrieben, existiert in jeder Partitionierungsstufe nur ein linearer Untercode $\mathcal{B}^{(i)}$, der die Code-Nullsequenz beinhaltet. Da $B^{(1)}$ ein linearer Faltungscode ist, der definitionsgemäß die Informations-Nullfolge auf eine Code-Nullfolge abbildet, lautet der *lineare Untercode* der i-ten Stufe bei der Partitionierung über die Informationsfolge

$$\mathcal{B}^{(i)} = \mathcal{B}^{(i)}_{\underline{0},\dots,\underline{0}}. \tag{3.14}$$

Das Nullsetzen der Informationsbits $x_\tau^{(1)}, \dots, x_\tau^{(i-1)}$ entspricht dem Streichen der ersten $(i-1)$ Zeilen der polynomialen Generatormatrix von $\mathcal{B}^{(1)}$ in Gleichung 2.64. Die verbleibende $(K^{(i)} \times N)$-Matrix ist daher die Generatormatrix des linearen Untercodes $\mathcal{B}^{(i)}$.

Über die freien Distanzen $d^{(i)}$ der Partitionierungsstufen kann man nur folgendes sagen:
Da die Partitionierung über die Informationsfolge unabhängig vom Code $\mathcal{B}^{(1)}$ erfolgt, tritt eine Distanzerhöhung von Stufe zu Stufe nur zufällig auf, daher der Name *zufällige Partitionierung*. Im allgemeinen wird keine optimale Partitionierung erreicht.
In Abschnitt 3.5 wird gezeigt, wie man das beschriebene Partitionierungsverfahren bezüglich der Distanzerhöhung verbessern kann.

3.3 Lineare und nichtlineare Untercodes

Dieser Abschnitt stellt eine grundlegende Überlegung zur Vereinfachung des weiteren Vorgehens dar. Es wird gezeigt, daß es genügt, die freie Distanz des linearen Untercodes $\mathcal{B}^{(i)}$ zu erhöhen, um die freien Distanzen aller Untercodes (also auch der nichtlinearen Untercodes oder „Cosets“ [1]) der i-ten Partitionierungsstufe zu erhöhen.

Hierzu sollen in diesem Abschnitt folgende Abkürzungen gelten:

$$\underline{a} = (\underline{a}^{(1)}, \ldots \underline{a}^{(j)}, \ldots, \underline{a}^{(i)}) \qquad \text{mit} \quad \underline{a}^{(j)} = (a_0^{(j)}, a_1^{(j)}, \ldots), \quad i < K \tag{3.15}$$

$$\underline{b} = (\underline{b}^{(1)}, \ldots, \underline{b}^{(j)}, \ldots, \underline{b}^{(K)}) \qquad \text{mit} \quad \underline{b}^{(j)} = (b_0^{(j)}, b_1^{(j)}, \ldots) \tag{3.16}$$

$$\underline{c} = (\underline{c}^{(1)}, \ldots, \underline{c}^{(j)}, \ldots, \underline{c}^{(K)}) \qquad \text{mit} \quad \underline{c}^{(j)} = (c_0^{(i)}, c_1^{(j)}, \ldots) \tag{3.17}$$

Geht man dann von einem linearen Faltungscode $\mathcal{B}^{(1)}$ aus, der wie oben beschrieben K-fach partitioniert wurde, so sind die Codefolgen $\underline{y}_{\underline{b}}$ und $\underline{y}_{\underline{c}}$ den Informationsfolgen $\underline{x}$ zugeordnet, für deren Teilfolgen $\underline{x}^{(i)} = \underline{b}^{(i)}$ bzw. $\underline{x}^{(i)} = \underline{c}^{(i)}$ gilt. Wählt man dabei

$$\underline{c}^{(j)} = \underline{b}^{(j)} = \underline{a}^{(j)} \neq \underline{0}, \qquad j = 1, \ldots, i \tag{3.18}$$

so gehören beide Codefolgen zum gleichen nichtlinearen Untercode $\mathcal{B}_{\underline{a}}^{(i+1)}$.

Da es sich bei $\mathcal{B}^{(1)}$ um einen linearen Code handelt, muß auch die Summe $\underline{y}_{\underline{b}} + \underline{y}_{\underline{c}}$ der beiden Codesequenzen eine gültige Codesequenz aus $\mathcal{B}^{(1)}$ darstellen. Die zugehörige Informationsfolge läßt sich aufgrund der Linearität ebenfalls durch Addition der beiden ursprünglichen Informationsfolgen ermitteln. Für die Numerierungen der Codefolge gilt dann wegen Gleichung 3.8:

$$\underline{z}^{(j)} \;=\; \underline{b}^{(j)} + \underline{c}^{(j)}, \qquad j = 1, \ldots, K \tag{3.19}$$

Insbesondere ergibt sich

$$\underline{z}^{(j)} = \underline{0}, \qquad j = 1, \ldots, i \tag{3.20}$$

das heißt, das durch die Addition entstandene Codewort gehört zum linearen Untercode:

$$\underline{y}_{\underline{b}} + \underline{y}_{\underline{c}} = \underline{y}_{\underline{b}+\underline{c}} \in \mathcal{B}^{(i+1)} \tag{3.21}$$

Das bedeutet aber, daß man allgemein jede beliebige Codefolge $\underline{y}_{\underline{b}}$ eines nichtlinearen Untercodes $\mathcal{B}^{(i+1)}_{\underline{z}^{(1)},\ldots,\underline{z}^{(i)}}$ durch Addition einer zweiten Codefolge des gleichen Codes auf eine Codefolge des linearen Codes $\mathcal{B}^{(i+1)}$ abbilden kann. Das heißt, die Decodierung jeder Codefolge kann auf die Decodierung im linearen Untercode $\mathcal{B}^{(i+1)}$ abgebildet werden (siehe auch Kapitel 4.2).

Daraus kann man schließen, daß die Codeeigenschaften aller Untercodes einer Partitionierungsstufe, insbesondere die freien Distanzen, identisch zu denen des linearen Untercodes sind. Es genügt daher, eine Partitionierung zu suchen, welche die freien Distanzen der linearen Untercodes maximiert. Im folgenden wird daher auch die *freie Distanz des linearen Untercodes* $\mathcal{B}^{(i)}$ mit $d^{(i)}$ bezeichnet (vgl Gleichung 3.4).

[1]siehe hierzu auch [Bos92], Seite 181

3.4 Pfadpunktierung bei zufälliger Partitionierung

In diesem Abschnitt soll veranschaulicht werden, was die oben beschriebene zufällige Partitionierung im Trellisdiagramm bewirkt. Die so gewonnene neue Betrachtungsweise wird dann in Abschnitt 3.5 eingesetzt, um eine bessere Partitionierung zu realisieren.

Eine besonders wichtige Rolle spielt dabei der Vergleich zwischen dem Trellis in Unit Memory Darstellung (UM-Trellis) und dem konventionellen Trellis. Zunächst wird der UM-Trellis betrachtet.

Hierzu werden die Gleichungen 2.66 und 2.67 benötigt. Mit Hilfe der Tatsache $\underline{z} = \underline{x}$, die für die zufällige Partitionierung mit Hilfe der Informationsfolge $\underline{x}$ nach Gleichung 3.11 gilt, sollen sie hier in etwas veränderter Form angegeben werden:

$$\underline{z}_{\tau-1} = \underline{\Gamma}_\tau \cdot \begin{pmatrix} \underline{0}_{[K]} \\ \underline{I}_{[K]} \end{pmatrix} \tag{3.22}$$

$$\underline{z}_{\tau-1} = \underline{\Theta}_\tau \tag{3.23}$$

Anschaulich sagen diese Gleichungen folgendes aus: Bei der Wahl eines Untercodes $\mathcal{B}^{(i)}_{\underline{z}^{(1)},\dots,\underline{z}^{(i-1)}}$ durch das Festlegen der Numerierungsfolgen $\underline{z}^{(1)},\dots,\underline{z}^{(i-1)}$ werden wegen Gleichung 3.23 gleichzeitig die ersten $(i-1)$ Bit jedes Zustands $\underline{\Theta}_\tau$ festgelegt. Dadurch wird gleichzeitig festgelegt, welche Zustände des UM-Trellis zu jedem Zeitpunkt τ möglich sind und welche nicht. Insbesondere gilt für die Wahl des linearen Untercodes $\mathcal{B}^{(i)}$

$$\underline{z}^{(1)} = \dots = \underline{z}^{(i-1)} = 0 \tag{3.24}$$

und damit

$$\underline{z}^{(j)}_{\tau-1} = \underline{\Theta}^{(j)}_\tau = 0, \qquad \text{für} \quad j = 1,\dots,(i-1). \tag{3.25}$$

Die Menge aller Codefolgen aus $\mathcal{B}^{(1)}$ wird somit aufgeteilt in solche, welche nur erlaubte Zustände nach Gleichung 3.25 durchlaufen, und andere. Die einen bilden den linearen Untercode $\mathcal{B}^{(i)}$, die anderen gehören zu einem der nichtlinearen Untercodes. Man kann auch sagen, der Code $\mathcal{B}^{(1)}$ geht durch die „*Punktierung*“ bestimmter Codefolgen in den linearen Untercode $\mathcal{B}^{(i)}$ über.

Bezeichnet man die Menge aller Zustände des UM-Trellis mit $\mathbf{V}$, so lautet die Menge der zulässigen Zustände im linearen Untercode $\mathcal{B}^{(i)}$ (unabhängig von $\mathcal{B}^{(1)}$):

$$\mathbf{V}_0^{(i)} = \Big\{ \quad \underline{\Theta}_\tau \in \mathbf{V} \quad \Big| \quad \underline{\Theta}_\tau = \left(0,\dots,0,z^{(i)}_{\tau-1},\dots,z^{(K)}_{\tau-1}\right); \quad z^{(i)}_{\tau-1},\dots,z^{(K)}_{\tau-1} \in \{0,1\} \quad \Big\}. \tag{3.26}$$

Es handelt sich dabei um die Menge $\{0,1,\dots,2^{K-i+1}-1\}$ der „oberen“ 2^{K-i+1} Zustände des UM-Trellis (siehe Bild 3.1, oben). Durch jeden weiteren Partitionierungsschritt wird die Anzahl dieser zulässigen Zustände halbiert.
Da die Menge $\mathbf{V}_0^{(i)}$ unabhängig vom Code $\mathcal{B}^{(1)}$ ist, findet keine gezielte Partitionierung bestimmter Codefolgen mit geringem Hamming-Gewicht statt. Die freie Distanz $d^{(i)}$ des Untercodes $\mathcal{B}^{(i)}$ wird daher in den meisten Fällen nicht größer sein als die des Codes $\mathcal{B}^{(1)}$.

Die Auswirkungen der zufälligen Partitionierung im konventionellen Trellis werden nun zunächst für zwei Spezialfälle betrachtet.

3.4.1 Pfadpunktierung durch Zustandspunktierung ($K = \nu$)

Als erstes soll betrachtet werden, welche Auswirkungen es hat, wenn man zur Beschreibung des zu partitionierenden Codes mit $k = 1$ und Einflußlänge ν den UM-Code der Dimension $K = \nu$ wählt. In Bild 3.2 auf Seite 26 sind hierzu für den Code $(5, 7)$ Ausschnitte aus dem UM-Trellis und darunter die entsprechenden Ausschnitte im konventionellen Trellis dargestellt. Im Beispiel erkennt man sofort, daß die Anzahl der Zustände im UM-Trellis mit der Anzahl der Zustände im konventionellen Trellis übereinstimmt. Dies gilt für den betrachteten Spezialfall wegen $K = \nu = m$ und $M = k = 1$ immer. Sowohl im konventionellen Trellis als auch im UM-Trellis existieren

$$2^{m \cdot k} = 2^{M \cdot K} = 2^{\nu} \tag{3.27}$$

Zustände. Die Definition eines Zustands im UM-Trellis lautet nach Gleichung 2.67 und 2.58:

$$\begin{aligned} \underline{\Theta}_{\tau} &= \underline{z}_{\tau-1} = (z_{\tau-1}^{(1)}, \ldots, z_{\tau-1}^{(K)}) \\ &= (u_{K\tau-K}, \ldots, u_{K\tau-1}) \end{aligned} \tag{3.28}$$

Einen Zustand im konventionellen Trellis kann man wegen Gleichung 2.36 wie folgt beschreiben:

$$\underline{\theta}_t = (u_{t-\nu}, \ldots, u_{t-1}) \tag{3.29}$$

Vergleicht man diese Darstellungen der Zustände, so erkennt man, daß sie für Zeitpunkte $t = K \cdot \tau$ identisch sind:

$$\boxed{\underline{\Theta}_{\tau} = \underline{\theta}_{K\tau}} \qquad \text{für } K = \nu \tag{3.30}$$

Das heißt jeder K-te Zustand des konventionellen Trellis ist identisch mit einem Zustand im UM-Trellis. Zustände, für die das gilt, werden im folgenden auch kurz als UM-Zustände bezeichnet. Sie sind für das Beispiel mit $\nu = 2$ in Tabelle 3.2 durch die graue Unterlegung gekennzeichnet.

Um die Darstellung dieses Zusammenhangs im weiteren zu vereinfachen, werden zwei neue Bezeichnungen eingeführt, die Zeit t_0 und die *Verschiebung* Δ:

$$t_0 := K \cdot \tau \qquad \Longleftrightarrow \qquad \Delta = 0 \tag{3.31}$$

Mit $t = t_0$ und $\Delta = 0$ werden in Zukunft die Zeitpunkte t bezeichnet, in denen Zustände im UM-Trellis und im konventionellen Trellis übereinstimmen. Dabei stellt die Verschiebung Δ eine Art periodische „Zeitachse“ dar (siehe Tabelle 3.2). Der Wert von Δ gibt die (zeitliche) Entfernung eines Zustands im konventionellen Trellis bis zum nächsten UM-Zustand an, also seine *Verschiebung*. Durch die Angabe von t_0 (bzw. τ) und Δ ist ein Zeitpunkt t eindeutig beschrieben, es gilt:

$$t = t_0 - \Delta. \tag{3.32}$$

Die Verschiebung Δ wird vor allem ab Kapitel 3.5.2 von großer Bedeutung sein.

Nach diesen Betrachtungen kann man die Überlegungen zur zufälligen Partitionierung aus dem vorigen Abschnitt relativ leicht auf den konventionellen Trellis übertragen.

UM-Code mit $K_0 = \nu = 2$ (4 Zustände)	Übergang	$\underline{\Gamma}_\tau$	0000		$00u_0u_1$		$u_0u_1u_2u_3$		
	Zustand	$\underline{\Theta}_\tau$	00		u_0u_1		u_2u_3		
	Info.vektor	$\underline{x}_\tau$	u_0u_1		u_2u_3		u_4u_5		
	Zeit	τ	0		1		2		...
	Verschiebung	Δ	0	1	0	1	0	1	...
Faltungscode mit $k = 1, \nu = 2$ (4 Zustände)	Zeit	t	0	1	2	3	4	5	...
	Übergang	$\underline{\gamma}_t$	000	$00u_0$	$0u_0u_1$	$u_0u_1u_2$	$u_1u_2u_3$	$u_2u_3u_4$	
	Zustand	$\underline{\theta}_t$	00	$0u_0$	u_0u_1	u_1u_2	u_2u_3	u_3u_4	
	Info.bit	u_t	u_0	u_1	u_2	u_3	u_4	u_5	

Tabelle 3.1: Darstellung eines Faltungscodes ($k = 1$) als UM-Code der Dimension $K = \nu$

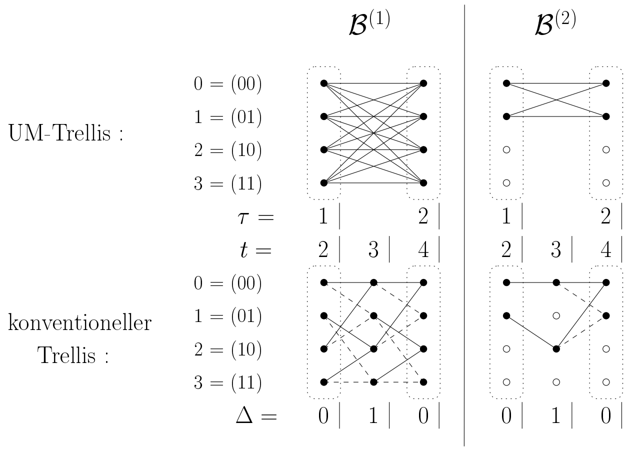


Abbildung 3.2: Zustandspunktierung im Trellis des Codes (5, 7) bei zufälliger Partitionierung 2. Ordnung

Gleichung 3.22 gilt natürlich weiterhin, doch kann man einen Übergang Γ_τ des UM-Trellis nun auch mit Hilfe zweier UM-Zustände des konventionellen Trellis beschreiben:

$$\underline{\Gamma}_\tau = (\underline{\theta}_{K(\tau-1)}, \underline{\theta}_{K\tau}) \tag{3.33}$$

Und Gleichung 3.23 geht für $t_0 = K \cdot \tau$ über in

$$\boxed{\underline{z}_{\tau-1} = \underline{\theta}_{t_0}.} \qquad \text{für } K = \nu \tag{3.34}$$

Das bedeutet, daß auch im konventionellen Trellis eine *Punktierung* von Zuständen bei der Wahl eines Untercodes $\mathcal{B}^{(i)}_{\underline{z}^{(1)},\ldots\underline{z}^{(i-1)}}$ stattfindet. Im weiteren soll daher von *Pfadpunktierung durch Zustandspunktierung* die Rede sein.

Im Unterschied zum UM-Trellis findet diese Zustandspunktierung im konventionellen Trellis jedoch nur zu Zeitpunkten $t_0 = K \cdot \tau$ statt, also zu Zeitpunkten mit der Verschiebung $\Delta = 0$. Für den linearen Untercode $\mathcal{B}^{(i)}$ lautet die Menge der zulässigen Zustände in diesen Zeitpunkten nach Gleichung 3.26:

$$\mathbf{V}_0^{(i)} = \Big\{ \ \underline{\theta}_{t_0} \in \mathbf{V} \ \Big| \ \underline{\theta}_{t_0} = \left(0, \ldots, 0, z_{\tau-1}^{(i)}, \ldots, z_{\tau-1}^{(K)}\right); \ z_{\tau-1}^{(i)}, \ldots, z_{\tau-1}^{(K)} \in \{0, 1\} \ \Big\}. \tag{3.35}$$

Die obigen Überlegungen zu den Elementen der Menge $\mathbf{V}_0^{(i)}$ sowie zu ihrer Mächtigkeit gelten hier unverändert. Beispiel 3.1 soll dies verdeutlichen.

Gewicht	Nr.	Pfad $\underline{\theta}$ mit den Zuständen $\underline{\theta}_t$ fett: UM-Zustände $\underline{\Theta}_\tau = \underline{\theta}_{\mathbf{t_0}}$								$\underline{\theta} \notin \mathcal{B}_{\underline{0}}^{(2)}$ $(\underline{\theta}_{t_0}) \notin \mathbf{V}_0^{(2)}$
5	1.1	**1**	2							
	1.2	**0**	1	**(2)**						×
6	2.1	**1**	2	**1**	2					
	2.2	**0**	1	**(2)**	1	**(2)**				×
	3.1	**1**	3	**(2)**						×
	3.2	**0**	1	**(3)**	2					×
7	4.1	**1**	2	1	**2**	1	2			
	4.2	**0**	1	**(2)**	1	**(2)**	1	**(2)**		×
	5.1	**1**	2	**1**	3	**(2)**				×
	5.2	**0**	1	**(2)**	1	**(3)**	2			×
	6.1	**1**	3	**(2)**	1	**(2)**				×
	6.2	**0**	1	**(3)**	2	**1**	2			×
	7.1	**1**	3	**(3)**	2					×
	7.2	**0**	1	**(3)**	3	**(2)**				×

Zeit	τ	**1**		**2**		**3**		**4**	
Verschieb.	Δ	**0**	1	**0**	1	**0**	1	**0**	1
Zeit	t	**2**	3	**4**	5	**6**	7	**8**	9

Tabelle 3.2: Codesequenzen des Codes $(5,7)$ als Zustandsfolgen

Beispiel 3.1:

Der Code $(5,7)$ besitzt die Gesamteinflußlänge $\nu = 2$. Wählt man zu seiner Beschreibung einen UM-Code der Dimension $K = \nu = 2$, so kann man wie oben beschrieben eine 2-fache Partitionierung über die Informationsfolge durchführen. Tabelle 3.1 zeigt die Darstellung des Codes in konventioneller Darstellung und in Unit Memory Darstellung. Die grau unterlegten Felder sollen die Übereinstimmung zwischen den UM-Zuständen $\underline{\Theta}_\tau$, den Zuständen $\underline{\theta}_{k\tau}$ und den Informationsvektoren $\underline{x}_{\tau-1} = \underline{z}_{\tau-1}$ verdeutlichen.

In Bild 3.2 sind kurze Ausschnitte aus dem UM-Trellis und dem konventionellen Trellis dargestellt. Wie man sieht, sind für den Ausgangscode $\mathcal{B}^{(1)}$ zu beliebigen Zeitpunkten alle Zustände zulässig. Im linearen Untercode $\mathcal{B}^{(2)}$ sind die UM-Zustände 2 und 3 punktiert, also nicht zulässig. Die Mengen der zulässigen Zustände in UM-Zeitpunkten $(\Delta = 0)$ lauten nach Gleichung 3.35:

$$
\begin{aligned}
\mathbf{V}_0^{(1)} &= \mathbf{V} = \{0,1,2,3\} \\
\mathbf{V}_0^{(2)} &= \left\{ \underline{\theta}_{t_0} \in \mathbf{V} \,\middle|\, \underline{\theta}_{t_0} = \left(0, z_{\tau-1}^{(2)}\right); \quad z_{\tau-1}^{(2)} \in \{0,1\} \right\} \\
&= \{(00),(01)\} \\
&= \{0,1\}
\end{aligned}
$$

Im UM-Trellis wird also kein Zustand aus $\overline{\mathbf{V}}_0^{(2)} = \{2,3\}$ von einer Codefolge $\underline{y} \in \mathcal{B}^{(2)}$ durchlaufen. Im konventionellen Trellis gilt diese Einschränkung für jeden zweiten Zeitpunkt $t_0 = 2\tau$.

In Tabelle 3.2 sind die Zustandsfolgen $\underline{\theta}$ aller Codefolgen des Codes $\mathcal{B}^{(1)} = (5,7)$ bis zum Hamming-Gewicht 7 aufgeführt. Die UM-Zustände sind dabei fett gedruckt. Durch die Klammern ( ) ist angedeutet, welche UM-Zustände im linearen Untercode $\mathcal{B}^{(2)}$ punktiert sind. Nur die Codefolgen, die keinen punktierten Zustand durchlaufen, gehören zu $\mathcal{B}^{(2)}$. Alle punktierten Codefolgen sind durch ein Kreuz in der letzten Spalte markiert, sie gehören zu einem der nichtlinearen Untercodes. Wie man sieht wird die Codefolge Nr. 1.1 nicht punktiert. Der lineare Untercode $\mathcal{B}^{(2)}$ besitzt daher die gleiche freie Distanz

$$
d^{(2)} = d^{(1)} = 5
$$

wie der Code $\mathcal{B}^{(1)}$. Hier wird also keine Distanzerhöhung durch die zufällige Partitionierung erzielt.

□

3.4.2 Pfadpunktierung durch Übergangspunktierung ($K = \nu + 1$)

Wählt man im Gegensatz zum vorhergehenden Abschnitt zur Beschreibung der Partitionierung den UM-Code der Dimension $K = \nu + 1$, so erhält man insgesamt eine Partitionierungsstufe mehr und die Auswirkungen im Trellisdiagramm sind etwas anders als oben:

Für $K = \nu + 1$ ist die Anzahl der Zustände in UM-Trellis um Faktor 2 größer als die Anzahl der Zustände im konventionellen Trellis. Sie entspricht damit der Anzahl der Übergänge im konventionellen Trellis (siehe auch Bild 3.3):

$$2^{(m+1)\cdot k} = 2^{M\cdot K} = 2^{\nu+1} \tag{3.36}$$

Motiviert durch diese Übereinstimmung kann man nun wie in Abschnitt 3.4.1 vorgehen und die Darstellung der UM-Zustände

$$\underline{\Theta}_\tau = (u_{K\tau-K}, \ldots, u_{K\tau-1})$$

mit der Darstellung der Übergänge $\underline{\gamma}_t$ des konventionellen Trellis nach Gleichung 2.36 vergleichen:

$$\underline{\gamma}_t = (u_{t-(\nu+1)}, \ldots, u_{t-1}) \tag{3.37}$$

Analog zum vorigen Abschnitt stellt man eine Übereinstimmung für $t = t_0 = K \cdot \tau$ fest:

$$\boxed{\underline{\Theta}_\tau = \underline{\gamma}_{K\tau}} \qquad \text{für } K = \nu + 1 \tag{3.38}$$

Das heißt, jeder K-te Übergang des konventionellen Trellis stimmt mit einem Zustand des UM-Codes der Dimension $K = \nu+1$ überein. Den Übergang $\underline{\Gamma}_\tau$ des UM-Trellis in Gleichung 3.22 kann man für diesen Fall also mit Hilfe zweier Übergänge des konventionellen Trellis darstellen, d.h. Gleichung 3.33 geht über in:

$$\underline{\Gamma}_\tau = (\underline{\gamma}_{K(\tau-1)}, \underline{\gamma}_{K\tau}) \tag{3.39}$$

Die für die Partitionierung entscheidende Gleichung 3.23 kann man dann darstellen als:

$$\boxed{\underline{z}_{\tau-1} = \underline{\gamma}_{t_0}.} \qquad \text{für } K = \nu + 1 \tag{3.40}$$

Sie besagt, daß bei $(v+1)$-facher Partitionierung eines Codes $\mathcal{B}^{(1)}$ im konventionellen Trellis durch das Festlegen der Untercode-Numerierung $\underline{z}^{(i)}$ nicht Zustände, sondern Übergänge zu bestimmten Zeitpunkten punktiert werden. Dies ist für $\nu = 2$ in Bild 3.3 dargestellt.
Wie im vorigen Abschnitt werden auch hier die Zeitpunkte t, zu denen solch eine Punktierung stattfindet, durch $\Delta = 0$ oder $t_0 = K \cdot \tau$ beschrieben. Man muß nur beachten, daß nun $K = \nu + 1$ gilt.

Auch durch die Punktierung von Übergängen werden bestimmte Pfade aus den Untercodes, insbesondere aus dem linearen Untercode, entfernt oder punktiert. Im folgenden soll dieser Effekt daher als *Pfadpunktierung durch Übergangspunktierung* bezeichnet werden.

Die Menge der zulässigen Übergänge des linearen Untercodes $\mathcal{B}^{(i)}$ zu den Zeitpunkten $t_0 = K \cdot \tau$ lautet für diesen Fall ganz analog zu Gleichung 3.35:

$$\mathbf{V}_0^{(i)} = \Big\{ \ \underline{\gamma}_{t_0} \in \mathbf{V} \ \Big| \ \underline{\gamma}_{t_0} = \left(0, \ldots, 0, z_{\tau-1}^{(i)}, \ldots, z_{\tau-1}^{(K)}\right); \quad z_{\tau-1}^{(i)}, \ldots, z_{\tau-1}^{(K)} \in \{0, 1\} \ \Big\}. \tag{3.41}$$

UM-Code mit $K_1 = \nu + 1 = 3$ (8 Zustände)	Übergang	$\underline{\Gamma}_\tau$	000000			$000u_0u_1u_2$			
	Zustand	$\underline{\Theta}_\tau$	000			$u_0u_1u_2$			
	Info.vektor	$\underline{x}_\tau$	$u_0u_1u_2$			$u_3u_4u_5$			
	Zeit	τ	0			1			...
	Verschiebung	Δ	0	2	1	0	2	1	...
Faltungscode mit $k = 1, \nu = 2$ (8 Übergänge)	Zeit	t	0	1	2	3	4	5	...
	Übergang	$\underline{\gamma}_t$	000	$00u_0$	$0u_0u_1$	$u_0u_1u_2$	$u_1u_2u_3$	$u_2u_3u_4$	
	Zustand	$\underline{\theta}_t$	00	$0u_0$	u_0u_1	u_1u_2	u_2u_3	u_3u_4	
	Info.bit	u_t	u_0	u_1	u_2	u_3	u_4	u_5	

Tabelle 3.3: Darstellung eines Faltungscodes ($k = 1$) als UM-Code der Dimension $K = \nu+1$

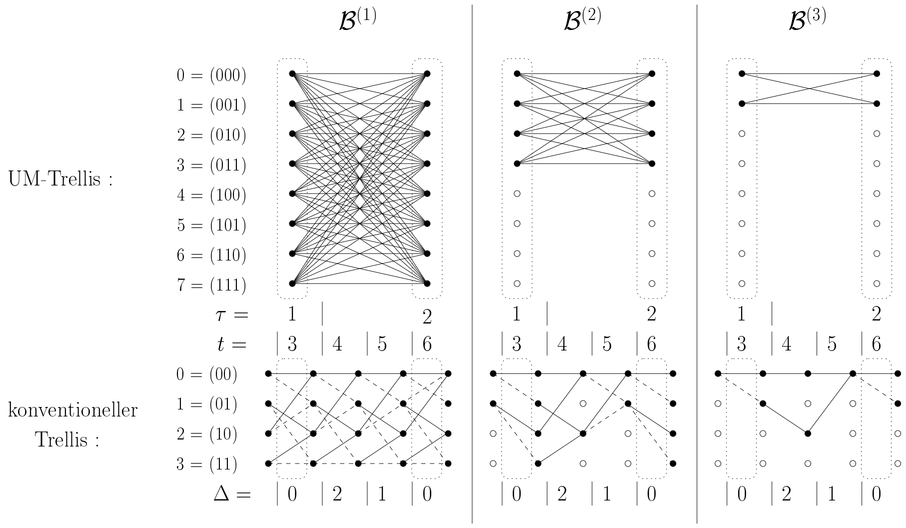


Abbildung 3.3: Übergangspunktierung im Trellis des Codes $(5, 7)$ bei zufälliger Partitionierung 3. Ordnung

Beispiel 3.2:

Stellt man den Code $(5, 7)$ aus Beispiel 3.1 wie in Tabelle 3.3 als UM-Code der Dimension $K = \nu+1 = 3$, so schafft man die Grundlage für eine Partitionierung 3-ter Ordnung über die Informationsfolge.

In Bild 3.3 sind Ausschnitte aus dem UM-Trellis und dem konventionellen Trellis der linearen Untercodes $\mathcal{B}^{(i)}$ abgebildet. Im UM-Trellis werden wie bereits in Beispiel 3.1 Zustände punktiert. Die Übergänge des konventionellen Trellis, die mit den UM-Zuständen übereinstimmen, liegen direkt darunter und sind auf gleiche Weise markiert. Man sieht, daß die Anzahl der zulässigen Zustände in den markierten Zeitpunkten $t_0 = 3\tau$ in jedem Partitionierungsschritt halbiert wird. Die Mengen der zulässigen Übergänge lassen sich mit Hilfe von Gleichung 3.41 berechnen:

$$
\begin{aligned}
\mathbf{V}_0^{(1)} &= \mathbf{V} = \{0,1,2,3,4,5,6,7\} \\
\mathbf{V}_0^{(2)} &= \left\{ \underline{\Theta}_\tau \in \mathbf{V} \;\middle|\; \underline{\Theta}_\tau = \left(0, z_{\tau-1}^{(2)}, z_{\tau-1}^{(3)}\right); \quad z_{\tau-1}^{(2)}, z_{\tau-1}^{(3)} \in \{0,1\} \right\} \\
&= \{(000),(001),(010),(011)\} \\
&= \{0,1,2,3\} \\
\mathbf{V}_0^{(3)} &= \left\{ \underline{\Theta}_\tau \in \mathbf{V} \;\middle|\; \underline{\Theta}_\tau = \left(0, 0, z_{\tau-1}^{(3)}\right); \quad z_{\tau-1}^{(3)} \in \{0,1\} \right\} \\
&= \{(000),(001)\} \\
&= \{0,1\}
\end{aligned}
$$

Gewicht	Nr.	Pfad $\underline{\gamma}$ mit den Übergängen $\underline{\gamma}_t$ fett: UM-Zustände $\underline{\Theta}_\tau = \underline{\gamma}_{t_0}$									$\underline{\gamma} \notin \mathcal{B}_0^{(2)}$ $(\gamma_{t_0}) \notin \mathbf{V}_0^{(2)}$	$\underline{\gamma} \notin \mathcal{B}_{0,0}^{(3)}$ $[\gamma_{t_0}] \notin \mathbf{V}_0^{(3)}$
5	1.1	**1**	2	4								
	1.2	**0**	1	2	(**4**)						×	×
	1.3	**0**	0	1	[**2**]	4						×
6	2.1	**1**	2	5	[**2**]	4						×
	2.2	**0**	1	2	(**5**)	2	4				×	×
	2.3	**0**	0	1	[**2**]	5	2	(**4**)			×	×
	3.1	**1**	3	6	(**4**)						×	×
	3.2	**0**	1	3	(**6**)	4					×	×
	3.3	**0**	0	1	[**3**]	6	4					×
7	4.1	**1**	2	5	[**2**]	5	2	(**4**)			×	×
	4.2	**0**	1	2	(**5**)	2	5	[**2**]	4		×	×
	4.3	**0**	0	1	[**2**]	5	2	(**5**)	2	4	×	×
	5.1	**1**	2	5	[**3**]	6	4					×
	5.2	**0**	1	2	(**5**)	3	6	(**4**)			×	×
	5.3	**0**	0	1	[**2**]	5	3	(**6**)	4		×	×
	6.1	**1**	3	6	(**5**)	2	4				×	×
	6.2	**0**	1	3	(**6**)	5	2	(**4**)			×	×
	6.3	**0**	0	1	[**3**]	6	5	[**2**]	4			×
	7.1	**1**	3	7	(**6**)	4					×	×
	7.2	**0**	1	3	(**7**)	6	4				×	×
	7.3	**0**	0	1	[**3**]	7	6	(**4**)			×	×

Zeit	τ	**1**			**2**			**3**		
Verschieb.	Δ	**0**	2	1	**0**	2	1	**0**	2	1
Zeit	t	**3**	4	5	**6**	7	8	**9**	10	11

Tabelle 3.4: Codesequenzen des Codes $(5,7)$ als Übergangsfolgen

Keiner der UM-Zustände aus $\overline{\mathbf{V}}_0^{(3)} = \{4,5,6,7\}$ wird von irgendeiner Codefolge des linearen Codes $\mathcal{B}^{(2)}$ durchlaufen und keiner der Zustände aus $\overline{\mathbf{V}}_0^{(3)} = \{2,3,4,5,6,7\}$ von einer Codefolge aus $\mathcal{B}^{(3)}$. Im konventionellen Trellis gilt diese Einschränkung für die Übergänge $\underline{\gamma}_t$ und zwar nur für jeden 3-ten Zeitpunkt $t = t_0 = 3\tau$ (siehe $t = 3, 6$ in Bild 3.3).

In Tabelle 3.4 sind schließlich wieder alle Codefolgen des Codes $(5,7)$ in UM-Darstellung bis zum Hamming-Gewicht 7 aufgelistet. Im Unterschied zu Tabelle 3.2 sind sie hier jedoch als Übergangsfolgen $\underline{\gamma}$ dargestellt. Mit runden Klammern ( ) sind alle Übergänge markiert, die im Code $\mathcal{B}^{(2)}$ bei zufälliger Partitionierung punktiert werden, und mit eckigen Klammern [ ] alle, die zusätzlich im Code $\mathcal{B}^{(3)}$ punktiert werden.
Wie in Beispiel 3.1 bleibt auch hier in allen linearen Untercodes ein Pfad vom Gewicht 5 unpunktiert, für die freien Distanzen gilt daher:

$$d^{(3)} = d^{(2)} = d^{(1)} = 5$$

Es wird keine Distanzerhöhung erreicht.

□

3.5 Optimale Partitionierung durch gezielte Punktierung

Im vorangegangenen Abschnitt wurde gezeigt, daß die Partitionierung eines Faltungscodes über die Informationsfolge nach Kapitel 3.2 nur in Ausnahmefällen Untercodes mit erhöhter freier Distanz zur Folge hat. Daher wird sie auch als zufällige Partitionierung bezeichnet. In diesem Abschnitt soll nun gezeigt werden, wie eine Partitionierung in Untercodes mit gleichzeitigem Anstieg der freien Distanz erzielt werden kann. Dabei wird schrittweise vorgegangen. In Abschnitt 3.5.1 wird zunächst an der Beschreibung eines Codes als UM-Code festgehalten und ein erstes Verfahren zur Distanzerhöhung im UM-Trellis beschrieben. Im Abschnitt 3.5.2 wird dieses Verfahren für den konventionellen Trellis verallgemeinert und gleichzeitig verbessert. Im letzten Abschnitt wird schließlich gezeigt, daß die Anzahl der Partitionierungsstufen prinzipiell unabhängig vom Partitionierungsverfahren gewählt werden kann. Man gelangt so schließlich zur allgemeinsten und besten Art der Partitionierung.

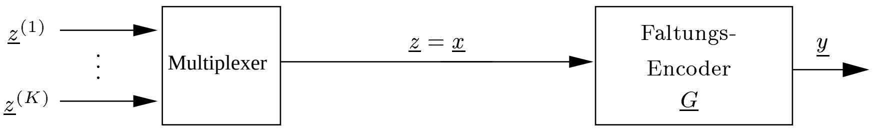


Abbildung 3.4: Multiplexer und Encoder bei zufälliger Partitionierung

Die Grundidee der zufälligen Partitionierung ist gleichzeitig ihr Problem: Die Informationsfolge (in UM-Darstellung), die zur Partitionierung eingesetzt wird, ist identisch zur Eingangsfolge des Encoders, durch die festgelegt wird, welche Zustände im Trellisdiagramm vorkommen (siehe Bild 3.4). Unabhängig vom betrachteten Faltungscode $\mathcal{B}^{(1)}$ werden in den linearen Untercodes immer dieselben Zustände des UM-Trellis punktiert (siehe Gleichung 3.26). Eine freie Wahl der zu punktierenden Zustände, so daß möglichst viele Pfade mit kleinem Gewicht aus den linearen Untercodes entfernt werden, ist nicht möglich.
Eine mögliche Lösung dieses Problems ist in Bild 3.5 dargestellt. Man unterscheidet nun zwei verschiedene Folgen: die Informations- bzw. Partitionierungsfolge $\underline{z}$ sowie die Eingangsfolge $\underline{x}$ des Encoders. Beide Folgen werden durch eine lineare, umkehrbare Abbildung miteinander verbunden, welche durch einen sogenannten Scrambler realisiert wird. Die Wahl eines Untercodes (z.B. des linearen Untercodes) erfolgt weiterhin, wie in Kapitel 3.2 beschrieben, mit Hilfe der Informationsfolge. Doch bietet die Wahl der linearen Abbildung nun (eingeschränkte) Möglichkeiten, festzulegen welche Zustände und welche Pfade dadurch punktiert werden sollen.

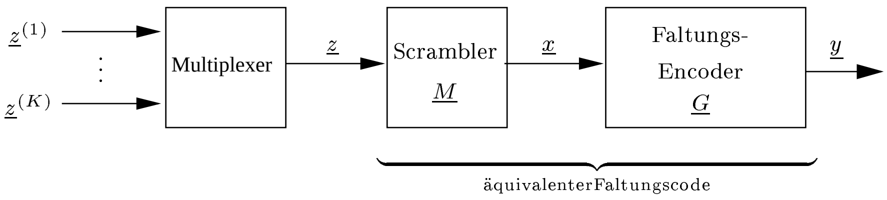


Abbildung 3.5: Multiplexer, Scrambler und Encoder bei gezielter Partitionierung

Mathematisch kann man einen Scrambler mit den Darstellungsmethoden von Faltungscodes (bzw. UM-Codes) beschreiben. Da durch einen Scrambler keine Redundanz hinzugefügt wird, soll er ab hier als Code der Rate 1 angesehen werden, er stellt jedoch keinen Code im allgemein üblichen Sinne dar.

Der Zusammenhang zwischen $\underline{z}$ und $\underline{x}$ lautet:

$$\underline{x} = \underline{z} \cdot \underline{M} \qquad \text{bzw.} \qquad \underline{z} = \underline{x} \cdot \underline{M}^{-1} \tag{3.42}$$

Dabei handelt es sich bei $\underline{M}$ (und bei $\underline{M}^{-1}$) um eine binäre halbunendliche Matrix nach Gleichung 2.16. Man kann natürlich auch die Polynomialschreibweise verwenden. Beachtet man dabei, daß es sich in der UM-Darstellung bei $\underline{x}(D)$ und $\underline{z}(D)$ um Vektoren der Länge K handelt, so muß die polynomiale Matrix $M(D)$ nach Gleichung 2.25 die Dimension $(K \times K)$ besitzen:

$$\underline{x}(D) = \underline{z}(D) \cdot \underline{M}(D) \tag{3.43}$$

Die Beschreibung mit Hilfe der inversen Scramblermatrix $\underline{M}(D)^{-1}$ ist dazu gleichwertig, wird aber in den folgenden Abschnitten von größerer Bedeutung sein als 3.43, da zunächst immer ein inverser Scrambler konstruiert wird:

$$\underline{z}(D) = \underline{x}(D) \cdot \underline{M}(D)^{-1} \tag{3.44}$$

Faßt man den Scrambler und den Encoder zusammen, so erhält man einen zum ursprünglichen Code *äquivalenten Faltungscode* (siehe [JW93]). Die Codesequenzen dieses Codes sind identisch zu denen des Ausgangscodes, doch ist die Zuordnung zwischen Informationsfolgen und Codefolgen verändert. Man spricht auch von einem veränderten Mapping. Ausgehend von der Generatormatrix $G(D)$ des ursprünglichen Codes in UM-Darstellung lautet die Generatormatrix des äquivalenten Codes:

$$\underline{G}'(D) = \underline{M}(D) \cdot \underline{G}(D) \tag{3.45}$$

Die Codefolge läßt sich somit direkt wie folgt berechnen:

$$\underline{y}(D) = \underline{x}(D) \cdot \underline{M}(D) \cdot \underline{G}(D) = \underline{z}(D) \cdot \underline{G}'(D) \tag{3.46}$$

Der Scrambler vor dem eigentlichen Faltungscode soll dazu eingesetzt werden, eine (möglichst) optimale Partitionierung zu realisieren. Das heißt, er soll benutzt werden, die freie Distanz des linearen Untercodes der jeweils betrachteten Partitionierungsstufe zu erhöhen. Eine Distanzerhöhung erreicht man, wenn es gelingt, alle Pfade zu punktieren, die als Hamming-Gewicht die alte freie Distanz besitzen.

In den folgenden Abschnitten soll nun für die Punktierung bestimmter Pfade durch Zustands- oder Übergangspunktierung folgende Schreibweise gelten:

Bisher wurde mit $\mathbf{V}_0^{(i)}$ die Menge der Zustände bezeichnet, die im UM-Trellis im linearen Untercode $\mathcal{B}^{(i)}$ zulässig sind. Analog dazu soll nun mit $\mathbf{V}_\Delta^{(i)}$ die Menge aller Zustände (oder Übergänge) bezeichnet werden, die im konventionellen Trellis zu Zeitpunkten $t = t_0 - \Delta$ im linearen Untercode $\mathcal{B}^{(i)}$ nicht punktiert werden, also zulässig sind. Wenn man mit $\mathbf{V}$ die Menge aller Zustände (Übergänge) des konventionellen Trellis bezeichnet, so ist

$$\overline{\mathbf{V}}_\Delta^{(i)} = \mathbf{V} \setminus \mathbf{V}_\Delta^{(i)} \tag{3.47}$$

die Komplementmenge zu $\mathbf{V}_\Delta^{(i)}$, das heißt die Menge der Zustände (oder Übergänge), die mit Hilfe des verwendeten Scramblers im linearen Untercode $\mathcal{B}^{(i)}$ punktiert werden.

Da im ursprünglichen Code $\mathcal{B}^{(1)}$ noch keine Punktierung erfolgt, gilt definitionsgemäß:

$$\overline{\mathbf{V}}_\Delta^{(1)} = \{\} \qquad \text{bzw.} \quad \mathbf{V}_\Delta^{(1)} = \mathbf{V} \qquad \text{für alle } \Delta \tag{3.48}$$

Mit $\overline{\mathbf{X}}_\Delta^{(i)}$ soll die Menge von Zuständen (oder Übergängen) bezeichnet werden, die man zu Zeitpunkten mit der Verschiebung Δ im linearen Untercode $\mathcal{B}^{(i)}$ (zusätzlich) punktieren möchte, um eine Distanzerhöhung gegenüber $\mathcal{B}^{(i-1)}$ zu erzielen. Wenn dies im Idealfall gelingt, gilt

$$\overline{\mathbf{X}}_\Delta^{(i)} \subseteq \overline{\mathbf{V}}_\Delta^{(i)}. \tag{3.49}$$

Die Bedeutung dieser Mengen soll in den folgenden Abschnitten noch weiter erläutert werden.

3.5.1 Blockscrambler (BS)

Die einfachste lineare Abbildung zwischen $\underline{z}$ und $\underline{x}$ ist sicher die blockweise Abbildung der Informationsvektoren $\underline{z}_\tau$ auf die Eingangsvektoren $\underline{x}_\tau$, die in [BDS96a] für den Fall der Zustandspunktierung beschrieben wird. Man erhält solch eine blockweise Abbildung, wenn man als inverse Scramblermatrix eine invertierbare binäre $(K \times K)$-Matrix $\underline{P}_0$ wählt:

$$\underline{M}(D)^{(-1)} = \underline{P}_0 \tag{3.50}$$

$$\underline{P}_0 = \left[\underline{p}^{(1)}, \dots, \underline{p}^{(K)}\right] \qquad \text{mit} \qquad \underline{p}^{(i)} = \begin{pmatrix} p_1^{(i)} \\ \vdots \\ p_l^{(i)} \end{pmatrix}, \quad l = K, \quad p_j^{(i)} \in \{0, 1\} \tag{3.51}$$

Man spricht in diesem Fall von einem (inversen) *Blockscrambler*, da man $\underline{P}_0$ prinzipiell als Generatormatrix eines Blockcodes der Rate 1 betrachten kann. Ein Blockcode entspricht aber einem Faltungscodes der Gedächtnislänge Null, es liegt also eine lineare Abbildung ohne Gedächtnis vor.
Der inverse Scrambler in halbunendlicher Schreibweise nach Gleichung 2.16 lautet:

$$\underline{M}^{-1} = \begin{bmatrix} \underline{P}_0 & & & 0 \\ & \underline{P}_0 & & \\ & & \underline{P}_0 & \\ 0 & & & \ddots \end{bmatrix} \tag{3.52}$$

Der Zusammenhang zwischen Informationsfolge $\underline{z}$ und Encodereingangsfolge $\underline{x}$ ist in Bild 3.6 auf Seite 34 dargestellt. Man kann ihn beschreiben durch

$$\underline{z}_\tau = \underline{x}_\tau \cdot \underline{P}_0, \tag{3.53}$$

und da weiterhin $\underline{x}_{\tau-1} = \underline{\Theta}_\tau$ gilt, geht Gleichung 3.23 über in

$$\underline{z}_{\tau-1} = \underline{\Theta}_\tau \cdot \underline{P}_0. \tag{3.54}$$

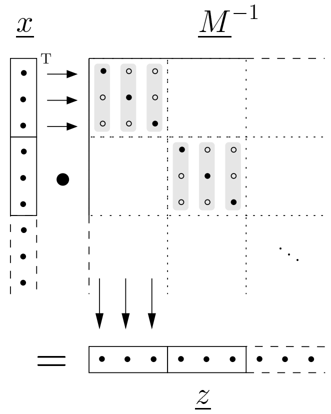


Abbildung 3.6: Abbildung von $\underline{x}$ auf $\underline{z}$ durch einen inversen Blockscrambler

Wählt man $\underline{P}_0$ ungleich der Einheitsmatrix, so bewirkt diese Abbildung, daß durch die Wahl der Informationsteilfolge $\underline{z}^{(i)} = \underline{0}$ andere Zustände im UM-Trellis des linearen Untercodes punktiert werden als bei der Partitionierung ohne Scrambler. Folgende Zustände bleiben übrig und werden von den Codefolgen des linearen Untercodes $\mathcal{B}^{(i)}$ durchlaufen:

$$\mathbf{V}_0^{(i)} = \Big\{ \quad \underline{\Theta}_\tau \in \mathbf{V} \quad \Big| \quad \underline{\Theta}_\tau \cdot \underline{P}_0 = \Big(0, \dots, 0, z_{\tau-1}^{(i)}, \dots, z_{\tau-1}^{(K)}\Big); \quad z_{\tau-1}^{(i)}, \dots, z_{\tau-1}^{(K)} \in \{0,1\} \quad \Big\} \tag{3.55}$$

Da jedes Element $z_\tau^{(i)}$ der partitionierenden Informationssequenz, wie in Bild 3.6 dargestellt, nur durch die i-te Spalte $\underline{p}^{(i)}$ der Matrix $\underline{P}_0$ mit dem Eingangsvektor $\underline{x}_\tau$ bzw. dem Zustand $\underline{\Theta}_{\tau+1}$ verbunden ist, kann man Gleichung 3.55 in i unabhängige Gleichungen aufspalten. Mit deren Hilfe kann man die Spalten der inversen Scramblermatrix so wählen, daß eine ganz bestimmte Menge $\overline{\mathbf{X}}_0^{(i+1)}$ von Zuständen im linearen Untercode $\mathcal{B}^{(i+1)}$ punktiert wird. Über die Gleichung

$$\mathbf{P}^{(i)} = \Big\{ \quad \underline{p}^{(i)} \quad \Big| \quad \underline{\Theta}_\tau \cdot \underline{p}^{(i)} = z_{\tau-1}^{(i)} \stackrel{!}{=} 1 \qquad \forall\, \underline{\Theta}_\tau \in \overline{\mathbf{X}}_0^{(i+1)} \quad \Big\} \tag{3.56}$$

erhält man eine Menge von möglichen Spaltenvektoren $\underline{p}^{(i)}$, die diese Bedingung erfüllen und man kann einen beliebigen von ihnen als i-te Spalte der Matrix $\underline{P}_0$ verwenden. Bei der Auswahl ist lediglich zu beachten, daß er linear unabhängig zu den bereits vorhandenen Spalten ist, um die Invertierbarkeit der Matrix $\underline{P}_0$ zu gewährleisten. Auf eine weitere Unterscheidung der möglichen Vektoren wird in 3.6.3 eingegangen.
Sollte die Menge $\mathbf{P}^{(i)}$ leer sein oder sollte keiner der Vektoren $\underline{p}^{(i)}$ linear unabhängig zu den bereits festgelegten Spalten $\underline{p}^{(1)}, \dots, \underline{p}^{(i-1)}$ sein, so ist die Menge $\overline{\mathbf{X}}_0^{(i+1)}$ kleiner zu wählen. Dies entspricht der Suche nach einem Vektor $\underline{p}^{(i)}$ der zwar nicht alle, aber möglichst viele der gewünschten Zustände punktiert.

Während der Konstruktion der Matrix $\underline{P}_0$ legt man also fest, welche UM-Zustände in den linearen Untercode punktiert werden. Im konventionellen Trellis entspricht dies je nach Wahl der Dimension K des UM-Codes der Punktierung bestimmter Zustände bzw. Übergänge zu den Zeitpunkten $t_0 = k\tau$.

Dementsprechend kann man auch die obigen Gleichungen mit den Elementen des konventionellen Trellis beschreiben. Man erhält:

- Für **Zustandspunktierung** $(K = \nu)$:

$$\left.\begin{array}{rcl} \underline{z}_{\tau-1} & = & \underline{\theta}_{t_0} \cdot \underline{P}_0 \\ z^{(i)}_{\tau-1} & = & \underline{\theta}_{t_0} \cdot \underline{p}^{(i)} \end{array}\right\} \quad \text{für} \quad t_0 = K \cdot \tau \tag{3.57}$$

$$\mathbf{P}^{(i)} = \left\{ \; \underline{p}^{(i)} \;\middle|\; \underline{\theta}_{t_0} \cdot \underline{p}^{(i)} = 1 \qquad \forall\, \underline{\theta}_{t_0} \in \overline{\mathbf{X}}_0^{(i+1)} \; \right\} \tag{3.58}$$

$$\mathbf{V}_0^{(i)} = \Big\{ \; \underline{\theta}_{t_0} \in \mathbf{V} \;\Big|\; \begin{array}{l} \underline{\theta}_{t_0} \cdot \underline{P}_0 = \left(0, \dots, 0, z^{(i)}_{\tau-1}, \dots, z^{(K)}_{\tau-1}\right); \\ z^{(i)}_{\tau-1}, \dots, z^{(K)}_{\tau-1} \in \{0,1\} \end{array} \Big\}. \tag{3.59}$$

- Für **Übergangspunktierung** $(K = \nu + 1)$:

$$\left.\begin{array}{rcl} \underline{z}_{\tau-1} & = & \underline{\gamma}_{t_0} \cdot \underline{P}_0 \\ z^{(i)}_{\tau-1} & = & \underline{\gamma}_{t_0} \cdot \underline{p}^{(i)} \end{array}\right\} \quad \text{für} \quad t_0 = K \cdot \tau \tag{3.60}$$

$$\mathbf{P}^{(i)} = \left\{ \; \underline{p}^{(i)} \;\middle|\; \underline{\gamma}_{t_0} \cdot \underline{p}^{(i)} = 1 \qquad \forall\, \underline{\gamma}_{t_0} \in \overline{\mathbf{X}}_0^{(i+1)} \; \right\} \tag{3.61}$$

$$\mathbf{V}_0^{(i)} = \Big\{ \; \underline{\gamma}_{t_0} \in \mathbf{V} \;\Big|\; \begin{array}{l} \underline{\gamma}_{t_0} \cdot \underline{P}_0 = \left(0, \dots, 0, z^{(i)}_{\tau-1}, \dots, z^{(K)}_{\tau-1}\right); \\ z^{(i)}_{\tau-1}, \dots, z^{(K)}_{\tau-1} \in \{0,1\} \end{array} \Big\}. \tag{3.62}$$

In den Beispielen 3.3 und 3.4 werden diese Gleichungen zur Scramblerkonstruktion angewandt.

Bevor in den folgenden Abschnitten weitere Scramblerarten beschrieben werden, soll noch der äquivalente Code betrachtet werden, der sich ergibt, wenn man den Scrambler und den ursprünglichen UM-Code zusammenfaßt. Auch bei ihm handelt es sich um einen UM-Code, da der Blockscrambler kein Gedächtnis besitzt.
Das Mapping zwischen Informationsbits und Übergängen im Trellis des ursprünglichen UM-Codes wird durch Gleichung 3.22 beschrieben und geht für den äquivalenten Code unter Berücksichtigung von Gleichung 3.54 über in:

$$\underline{z}_{\tau-1} = \underline{\Gamma}_\tau \cdot \begin{bmatrix} \underline{0}_{[K]} \\ \underline{P}_0 \end{bmatrix} \tag{3.63}$$

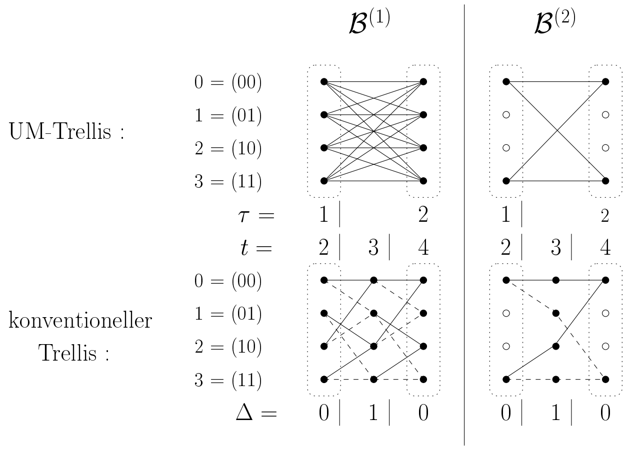


Abbildung 3.7: Zustandspunktierung im Trellis des Codes $(5,7)$ bei 2-facher Partitionierung mit einem Blockscrambler

Die zufällige Partitionierung nach Kapitel 3.2 stellt also, wie bereits oben erwähnt, einen Sonderfall der Partitionierung mit Hilfe eines Blockscramblers dar, für den $\underline{P}_0 = \underline{I}_{[K]}$ gewählt wird:

$$\underline{M}(D)^{-1} = \underline{I}_{[K]} \tag{3.64}$$

Da diese Wahl bei der zufälligen Partitionierung unabhängig vom Code $\mathcal{B}^{(1)}$ erfolgt, erzielt man mit ihr nur in Ausnahmefällen eine Distanzerhöhung. Erst durch Anpassung des Scramblers an den Code kann dies erreicht werden.

Beispiel 3.3:

Betrachtet man Tabelle 3.2 aus Beispiel 3.1, so erkennt man, daß beide Pfade von Gewicht 5 punktiert werden können, wenn es möglich ist, die UM-Zustände $\underline{\theta}_{t_0} = 1$ und $\underline{\theta}_{t_0} = 2$ zu punktieren. Man wählt daher $\underline{\mathbf{X}}_0^{(2)} = \{1,2\} = \{(01),(10)\}$ und erhält über Gleichung 3.56 bzw. 3.58 die Menge $\mathbf{P}^{(1)} = \{(11)^T)\}$. Das heißt es existiert genau ein Vektor $\underline{p}^{(1)} = (11)^T$, der die gewünschten UM-Zustände punktiert, für den also $\mathbf{V}_0^{(1)} = \{0,3\}$ gilt. Dieser wird als erste Spalte der inversen Scramblermatrix verwendet und man erzielt dadurch, daß der lineare Untercode $\mathcal{B}^{(2)}$ eine freie Distanz $\mathrm{d}^{(2)} > 5$ besitzt. Da der Pfad 3.2 mit dem Hamming-Gewicht 6 keine punktierten UM-Zustände durchläuft und somit nicht punktiert wird, ergibt sich $\mathrm{d}^{(2)} = 6$.

Die zweite Spalte der Matrix $\underline{P}_0$ kann prinzipiell beliebig gewählt werden, muß aber linear unabhängig von $\underline{p}^{(1)}$ sein. Sie hat keine Auswirkung auf die Partitionierung. Man kann beispielsweise $\underline{p}^{(2)} = (01)^T$ wählen und erhält die inverse Scramblermatrix

$$\underline{M}(D)^{-1} = \underline{P}_0 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Da diese Matrix die Determinante 1 besitzt, ist der resultierende Scrambler $\underline{M}(D)$ nichtrekursiv (siehe Kapitel 2.1.3).
Bild 3.7 zeigt einen Ausschnitt aus dem Trellis von $\mathcal{B}^{(2)}$ mit den punktierten Zuständen. Ein Vergleich mit Bild 3.2 auf Seite 3.2 zeigt den Unterschied zur Punktierung bei zufälliger Partitionierung.

□

Beispiel 3.4:

Zur Konstruktion eines Blockscramblers für 3-fache Partitionierung des Codes $(5,7)$ wird Tabelle 3.4 von Beispiel 3.2 betrachtet. Damit die freie Distanz im ersten Partitionierungsschritt erhöht wird,

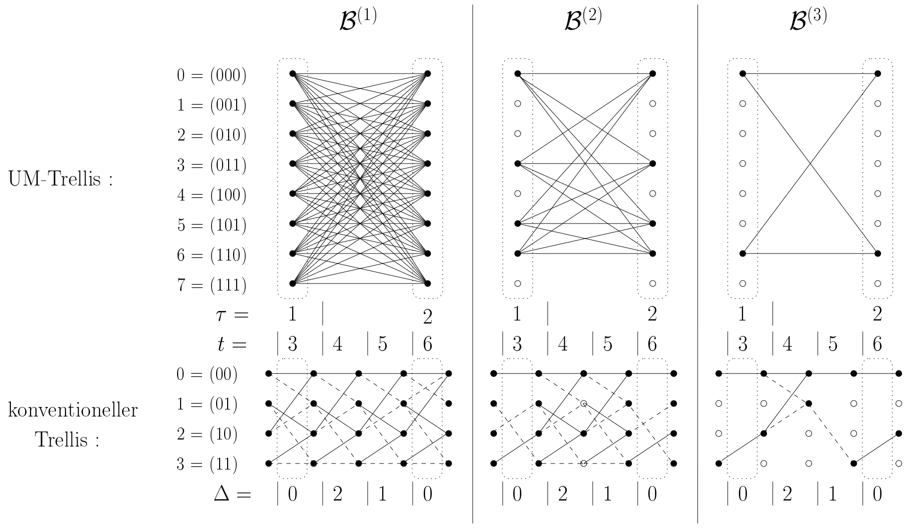


Abbildung 3.8: Übergangspunktierung im Trellis des Codes $(5,7)$ bei 3-facher Partitionierung mit einem Blockscrambler

muß muß man zu den Zeitpunkten $t_0 = K\tau$ auf jeden Fall die Übergänge $\overline{\mathbf{X}}_0^{(2)} = \{1,2,4\}$ punktieren. Mit Hilfe von Gleichung 3.61 erhält man $\mathbf{P}^{(2)} = \{(111)^T)\}$, d.h. es existiert wiederum nur ein Vektor, der die gegebene Bedingung erfüllt. Durch ihn wird außer den gewünschten Übergängen auch noch der Übergang 7 punktiert. Dadurch werden weitere Pfade mit höherem Gewicht entfernt, z.B der Pfad 7.1. Dagegen können die Pfade 2.1, 3.1 und 3.2 vom Gewicht 6 im ersten Partitionierungsschritt nicht punktiert werden. Es ergibt sich daher die freie Distanz $d^{(2)} = 6$.

Bei der Wahl der zweiten Spalte des inversen Scramblers ist das Ziel wiederum die Erhöhung der freien Distanz. Es sollen jetzt möglichst alle verbliebenen Pfade vom Gewicht 6 entfernt werden. Nach Tabelle 3.4 müßte man hierzu die Menge der zu punktierenden Pfade als $\overline{\mathbf{X}}_\tau^{(3)} = \{3,5,6\}$ wählen. Man stellt bei Anwendung von Gleichung 3.61 jedoch fest, daß kein Vektor $\underline{p}^{(2)}$ existiert, der alle diese Zustände punktiert. Anschaulich kann man dies damit begründen, daß im Untercode $\mathcal{B}^{(3)}$ auch bei gezielter Partitionierung zu jedem Zeitpunkt mit $\Delta = 0$ noch genau 2 Zustände zulässig sein müssen (vgl. Bild 3.2). Man verkleinert daher die Menge $\mathbf{X}_0^{(3)}$ (z.B. auf $\{3,5\}$), und wählt somit als zweite Spalte einen Vektor $\underline{p}^{(2)}$, der nur zwei der gewünschten Übergänge punktiert. Aus mehreren Möglichkeiten wählt man eine, bei der auch möglichst viele Pfade mit größerem Gewicht entfernt werden, $\underline{p}^{(2)} = (001)^T$ erfüllt diese Forderungen, es ergibt sich $\overline{\mathbf{V}}_0^{(2)} = \{1,2,4,7\} \cup \{1,3,5,7\} = \{1,2,3,4,5,7\}$. Die freie Distanz kann in dieser Partitionierungsstufe nicht weiter erhöht werden, da Pfad 3.2 nicht punktiert wird.

Die letzte Spalte wählt man möglichst so, daß sich für die Determinante der inversen Scramblermatrix der Wert 1 ergibt (vgl. Abschnitt 3.6.3) und erhält beispielsweise die inverse Scramblermatrix

$$\underline{M}(D)^{-1} = \underline{P}_0 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Die freien Distanzen der Untercodes lauten bei Partitionierung mit diesem Scrambler

$$d^{(3)} = d^{(2)} = 6.$$

Zum Vergleich mit der zufälligen Partitionierung in Bild 3.3 sind die Auswirkungen der gezielten Übergangspunktierung im Trellis in Bild 3.8 dargestellt.

□

3.5.2 Unit Memory Scrambler (UMS)

Im vorigen Abschnitt wurde die gezielte Punktierung bestimmter Pfade analog zur zufälligen Partitionierung durch die Punktierung bestimmter UM-Zustände realisiert. Diese sind bei geeigneter Wahl von K identisch zu Zuständen bzw. Übergängen des konventionellen Trellis (zu Zeitpunkten $t_0 = K\tau$ mit der Verschiebung $\Delta = 0$).
An dieser Stelle soll nun untersucht werden, wie sich die Partitionierung auf die Zustände und Übergänge des konventionellen Trellis auswirkt, welche nicht mit Zuständen im UM-Trellis übereinstimmen ($\Delta \neq 0$). Dazu sei noch einmal Bild 3.2 betrachtet. Die zufällige Partitionierung hat im linearen Untercode $\mathcal{B}^{(2)}$ nicht nur die Punktierung der Zustände 2 und 3 zu den Zeitpunkten $t = 0, 2, 4, \dots$ (also $\Delta = 0$) zur Folge, sondern bewirkt gleichzeitig eine Punktierung der Zustände 1 und 3 zu den Zeitpunkten $t = 1, 3, \dots$ ($\Delta = 1$). Auch in Tabelle 3.2 kann man dies ablesen:
Das Informationsbit u_0 ist nicht nur im Zustand $\underline{\theta}_2$ enthalten, sondern auch in $\underline{\theta}_1$. Wird es mit der Partitionierungsfolge $\underline{z}^{(1)} = \underline{x}^{(1)}$ zu Null gesetzt, so bedeutet dies auch eine Punktierung bestimmter Zustände zum Zeitpunkt $t = 1$.

Allgemein wird dieser Zusammenhang von Gleichung 2.36 beschrieben. Für $k = 1$ und $m = \nu$ ergibt sich

$$\underline{\gamma}_{t+1} = (\underbrace{u_{t-\nu}, \overbrace{u_{t-\nu+1}, \dots, u_{t-1}}^{}}_{\underline{\theta}_t} \! \phantom{u_{t-\nu},}\overbrace{\phantom{u_{t-\nu+1}, \dots, u_{t-1},}\; u_t}^{\underline{\theta}_{t+1}}) \qquad (3.65)$$

Jedes Informationsbit u_t hat demnach Einfluß auf die ν Zustände $\underline{\theta}_{t+1}, \dots, \underline{\theta}_{t+\nu+1}$ und auf die $\nu + 1$ Übergänge $\underline{\gamma}_t, \dots, \underline{\gamma}_{t+\nu+1}$. Wählt man also wie oben zur Beschreibung den UM-Code der Dimension $K = \nu$ oder $K = \nu + 1$, so bewirkt die Wahl der Partitionierungsfolge $\underline{z}^{(i)}$, also das Festlegen jedes K-ten Bits, eine Partitionierung bestimmter Zustände (bzw. Übergänge) zu *allen* möglichen Zeitpunkten t (mit beliebiger Verschiebung $0 \leq \Delta < K - 1$) und nicht nur zu jedem K-ten Zeitpunkt t_0 mit $\Delta = 0$.

Auch im Beispiel nach Bild 3.3 kann man dies gut erkennen: Im konventionellen Trellis werden im linearen Untercode $\mathcal{B}^{(2)}$ beispielsweise zu den Zeitpunkten $t = 0, 3, 6, \dots$ (mit $\Delta = 0$) die Übergänge 1,3,5,7 punktiert, da deren letztes Bit ungleich 0 ist. Zu den Zeitpunkten 1,4,7, ... ($\Delta = 1$) werden die Übergänge 2,3,6 und 7 punktiert, da das mittlere Bit bei ihnen ungleich 0 ist.

Punktiert man in der UM-Zeitpunkten ($\Delta = 0$) andere Zustände bzw. Übergänge, z.B. mit Hilfe eines Blockscramblers, so hat dies andere Auswirkungen auf die restlichen Zustände bzw. Übergänge. Dies kann man in den Bildern 3.7 und 3.8 erkennen: In den „Zwischen-Zeitpunkten“ mit $\Delta \neq 0$ findet nur teilweise eine Punktierung statt.

Diese Tatsache soll zur gezielten Pfadpunktierung genutzt werden, indem man im konventionellen Trellis zu allen möglichen Zeitpunkten eine gezielte Punktierung bestimmter Zustände oder Übergänge durchführt.

Hierbei wird zunächst wieder ein Sonderfall betrachtet:

Unit Memory Scrambler in Diagonalform (DS)

Dieser Sonderfall wird in [BDS96b] und [Die96] beschrieben, der resultierende Scrambler wird dort als Convolutional Scrambler bezeichnet. Da in der vorliegenden Arbeit jedoch die allgemeinste Form eines Scramblers so bezeichnet wird (Faltungsscrambler, CS), soll dieser Spezialfall hier als *Diagonalscrambler* (DS) oder *UM-Scrambler in Diagonalform* bezeichnet werden. Der Grund für diese Bezeichnung liegt in der Struktur der inversen Scramblermatrix (s.u.).

Punktiert man in jeder Partitionierungsstufe jeweils zu Zeitpunkten mit unterschiedlicher Verschiebung Δ, so hat man mehr Freiheit bei der Wahl der zu punktierenden Zustände oder Übergänge. Zunächst sollen im ersten Partitionierungsschritt Zustände oder Übergänge zu den Zeitpunkten mit der Verschiebung $\Delta = (K-1)$ punktiert werden, im nächsten Schritt zu den Zeitpunkten mit $\Delta = (K-2)$ usw. Das heißt, die i-te Informationsfolge $\underline{z}^{(i)}$ punktiert jeweils Zustände oder Übergänge zu Zeitpunkten, die gegenüber UM-Zuständen um

$$\Delta^{(i)} = K - i \tag{3.66}$$

verschoben sind. Man unterscheidet wieder die beiden Punktierungsarten:

- **Zustandspunktierung** $(K = \nu)$

  Gleichung 3.57 die den Zusammenhang zwischen Zuständen des konventionellen Trellis und der Informationssequenz beschreibt, geht für den Diagonalscrambler über in:

  $$z^{(i)}_{\tau-1} = \underline{\theta}_t \cdot \underline{p}^{(i)} \qquad \text{für} \quad t = K\tau - (K-i) \tag{3.67}$$

  Die Menge der Vektoren, die alle gewünschten Zustände $\overline{\mathbf{X}}^{(i+1)}_{\Delta^{(i+1)}}$ punktieren, wird prinzipiell wie in Gleichung 3.58 berechnet:

  $$\mathbf{P}^{(i)} \;=\; \left\{ \;\underline{p}^{(i)} \;\middle|\; \underline{\theta}_t \cdot \underline{p}^{(i)} = 1 \qquad \forall\, \underline{\theta}_t \in \overline{\mathbf{X}}^{(i+1)}_{K-(i+1)} \;\right\} \tag{3.68}$$

  Und die Menge der Zustände, die durch einen bestimmten Vektor $\underline{p}^{(i-1)}$ in allen Zeitpunkten mit Verschiebung $\Delta^{(i)} = K - i$ im linearen Untercode $\mathcal{B}^{(i)}$ punktiert werden, geht aus Gleichung 3.59 hervor:

  $$\overline{\mathbf{V}}^{(i)}_{K-i} \;=\; \left\{ \;\underline{\theta}_t \in \mathbf{V} \;\middle|\; \underline{\theta}_t \cdot \underline{p}^{(i-1)} = 1 \;\right\} \tag{3.69}$$

- **Übergangspunktierung** $(K = \nu + 1)$

  Die Gleichungen für die Übergangspunktierung mit Diagonalscrambler lauten völlig analog:

  $$z^{(i)}_{\tau} = \underline{\gamma}_t \cdot \underline{p}^{(i)} \qquad \text{für} \quad t = K\tau - (K-i) \tag{3.70}$$

  $$\mathbf{P}^{(i)} \;=\; \left\{ \;\underline{p}^{(i)} \;\middle|\; \underline{\gamma}_t \cdot \underline{p}^{(i)} = 1 \qquad \forall\, \underline{\gamma}_t \in \overline{\mathbf{X}}^{(i+1)}_{K-(i+1)} \;\right\} \tag{3.71}$$

  $$\overline{\mathbf{V}}^{(i)}_{K-i} \;=\; \left\{ \;\underline{\gamma}_t \in \mathbf{V} \;\middle|\; \underline{\gamma}_t \cdot \underline{p}^{(i-1)} = 1 \;\right\} \tag{3.72}$$

  Ein Beispiel zur Anwendung der Gleichungen findet sich auf Seite 41.

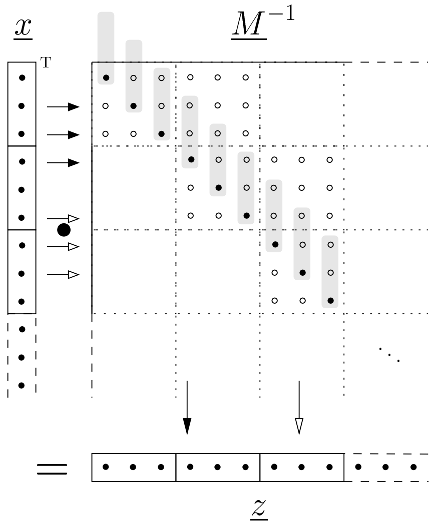


Abbildung 3.9: Abbildung von $\underline{z}$ auf $\underline{x}$ durch einen inversen Diagonalscrambler

Das Vorgehen bei der Konstruktion eines Diagonalscramblers ist prinzipiell das gleiche wie bei der eines Blockscramblers, nur daß man in jeder Stufe die Zustände oder Übergänge in den zu punktierenden Pfaden zu unterschiedlichen Zeitpunkten t (Verschiebungen $\Delta^{(i)}$) betrachten muß (siehe Beispiel 3.5).

Der Zusammenhang zwischen Informationsfolge $\underline{z}$ und Encodereingangsfolge $\underline{x}$ der durch die Gleichungen 3.67 und 3.70 beschrieben wird, ist in Bild 3.9 für $K = 3$ dargestellt. Dabei ist die Abhängigkeit der einzelnen Informationsbits von je K Bits der Encodereingansfolge, die die Zusande $\underline{\theta}_t$ oder Übergänge $\underline{\gamma}_t$ im konventionellen Trellis bilden, durch die grauen Bereiche gekennzeichnet. Diese entsprechen den Vektoren $\underline{p}^{(i)}$.

Um die gewünschte Partitionierung über eine inverse Scramblermatrix zu beschreiben, muß man die mit Hilfe von Gleichung 3.68 bzw. 3.71 bestimmten Vektoren $\underline{p}^{(i)}$ zu einer Matrix $\underline{P}$ anordnen, wie in Bild 3.10 dargestellt.

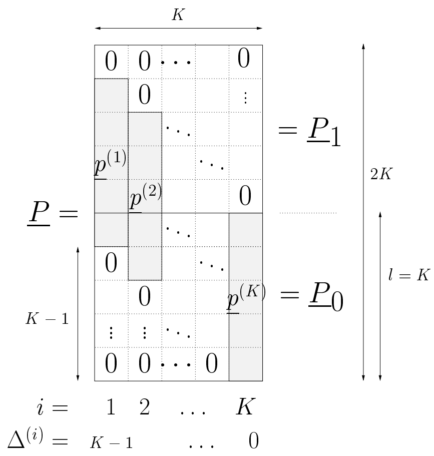


Abbildung 3.10: Teilmatrizen eines inversen Diagonalscramblers

Jeder Vektor $\underline{p}^{(i)}$ ist in dieser Matrix um $\Delta = K - i$ Stellen nach oben verschoben. Alle Matrixelemente, die nicht von einem der Vektoren belegt sind, sind Null.

Entsprechend Bild 3.9 setzt sich die halbunendliche inverse Scramblermatrix dann aus den Teilmatrizen $\underline{P}_0$ und $\underline{P}_1$ folgendermaßen zusammen:

$$\underline{M}^{-1} = \begin{bmatrix} \underline{P}_0 & \underline{P}_1 & & & 0 \\ & \underline{P}_0 & \underline{P}_1 & & \\ & & \underline{P}_0 & \underline{P}_1 & \\ 0 & & & & \ddots \end{bmatrix} \tag{3.73}$$

Diese besitzt die Form der Generatormatrix eines Unit Memory Codes (vgl. 2.55), daher werden die so konstruierten Scrambler als *UM-Scrambler* (UMS) bezeichnet. Aufgrund der Anordnung der Vektoren $\underline{p}^{(i)}$ wird dieser Spezialfall, wie bereits oben erwähnt, auch *Diagonalscrambler* (DS) genannt.

Nach Gleichung 2.64 läßt sich sofort die polynomiale inverse Scramblermatrix angeben, die wie bei einem UM-Code Gedächtnislänge 1 hat:

$$\underline{M}(D)^{-1} = \underline{P}_0 + D \cdot \underline{P}_1 \tag{3.74}$$

und durch die folgende Abbildung beschrieben wird:

$$\underline{z}_\tau = \underline{x}_{\tau-1} \cdot \underline{P}_1 + \underline{x}_\tau \cdot \underline{P}_0 \tag{3.75}$$

Wegen $\underline{x}_\tau = \underline{\Theta}_{\tau+1}$ und Gleichung 2.65 kann man diesen Zusammenhang auch schreiben als

$$\underline{z}_{\tau-1} = \underline{\Gamma}_\tau \cdot \begin{bmatrix} \underline{P}_1 \\ \underline{P}_0 \end{bmatrix} = \underline{\Gamma}_\tau \cdot \underline{P} \ , \tag{3.76}$$

und ist damit wieder bei einer Beschreibung nach Gleichung 3.22 angelangt, die die Zuordnung zwischen Informationsbits und Übergängen im UM-Trellis beschreibt. Gegenüber dem Mapping beim Blockscrambler (Gleichung 3.63) wurde dabei eine weitere Verallgemeinerung vorgenommen, indem die Nullmatrix durch $\underline{P}_1$ ersetzt wurde.

Angesichts der schrittweisen Verallgemeinerung, die zu dieser Gleichung geführt hat, stellt sich die Frage, ob noch weitere Verallgemeinerungen möglich und sinnvoll sind. Dies wird im folgenden Abschnitt beantwortet. Zuvor soll jedoch noch ein Beispiel betrachtet werden.

Beispiel 3.5:

Der Code $(5,7)$ soll mit Hilfe eines Diagonalscramblers zur Übergangspunktierung partitioniert werden. Wie in Beispiel 3.3 auf Seite 36 soll also eine 3-fach Partitionierung erfolgen. Wählt man die Vektoren $\underline{p}^{(1)} = (111)^T$ und $\underline{p}^{(2)} = (011)^T$, so werden in Zeitpunkten t mit $\Delta = 3-1 = 2$ die Übergänge $\overline{\mathbf{V}}_2^{(2)} = \{1,2,4,7\}$ und zu Zeitpunkten mit $\Delta = 3-2 = 1$ die Übergänge $\overline{\mathbf{V}}_1^{(3)} = \{1,2,5,6\}$ punktiert. Wählt man die letzten Spalte geeignet und fügt alle Spalten wie oben beschrieben zur Matrix $\underline{P}$ zusammen, so ergibt sich beispielsweise

$$\underline{P} = \begin{bmatrix} 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{0} & 0 \\ & & \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \\ 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} \end{bmatrix} \quad \text{und} \quad \underline{M}(D)^{-1} = \begin{bmatrix} 1 & 1 & 0 \\ D & 1 & 1 \\ D & 0 & 1 \end{bmatrix}$$

Gewicht	Nr.	Pfad $\underline{\gamma}$ mit den Übergängen $\underline{\gamma}_t$ fett: UM-Zustände $\underline{\Theta}_\tau$									$\underline{\gamma} \notin \mathcal{B}_0^{(2)}$ $(\gamma_{t_0-2}) \notin \mathbf{V}_2^{(2)}$	$\underline{\gamma} \notin \mathcal{B}_{0,0}^{(3)}$ $[\gamma_{t_0-1}] \notin \mathbf{V}_1^{(3)}$
5	1.1	**1**	(2)	4							×	×
	1.2	**0**	(1)	[2]	**4**						×	×
	1.3	**0**	0	[1]	**2**	(4)					×	×
6	2.1	**1**	(2)	[5]	**2**	(4)					×	×
	2.2	**0**	(1)	[2]	**5**	(2)	4				×	×
	2.3	**0**	0	[1]	**2**	5	[2]	**4**				×
	3.1	**1**	3	[6]	**4**							×
	3.2	**0**	(1)	3	**6**	(4)					×	×
	3.3	**0**	0	[1]	**3**	6	4					×
7	4.1	**1**	(2)	5	**2**	5	[2]	**4**			×	×
	4.2	**0**	(1)	[2]	**5**	(2)	[5]	**2**	(4)		×	×
	4.3	**0**	0	[1]	**2**	5	[2]	**5**	(2)	4	×	×
	5.1	**1**	(2)	5	**3**	6	4				×	×
	5.2	**0**	(1)	[2]	**5**	3	[6]	**4**			×	×
	5.3	**0**	0	[1]	**2**	5	3	**6**	(4)		×	×
	6.1	**1**	3	[6]	**5**	(2)	4				×	×
	6.2	**0**	(1)	3	**6**	5	[2]	**4**			×	×
	6.3	**0**	0	[1]	**3**	6	[5]	**2**	(4)		×	×
	7.1	**1**	3	7	**6**	(4)					×	×
	7.2	**0**	(1)	3	**7**	6	4				×	×
	7.3	**0**	0	[1]	**3**	7	[6]	**4**				×
8	8.1	**1**	3	7	**7**	6	4					
	8.2	**0**	(1)	3	**7**	(7)	[6]	**4**			×	×
	8.3	**0**	0	[1]	**3**	(7)	7	**6**	4		×	×
	...					...						

Zeit	τ	**1**			**2**			**3**		
Verschieb.	Δ	**0**	2	1	**0**	2	1	**0**	2	1
Zeit	t	**3**	4	5	**6**	7	8	**9**	10	11

Tabelle 3.5: Codesequenzen des Codes $(5, 7)$ als Übergangsfolgen, Pfadpunktierung mit Diagonalscrambler

Die punktierten Übergänge sind in Tabelle 3.5 durch die Klammern ( ) bzw. [ ] markiert. In $\mathcal{B}^{(3)}$ und $\mathcal{B}^{(2)}$ punktierte Pfade sind durch ein Kreuz in der letzten bzw. vorletzten Spalte gekennzeichnet. Man erzielt mit dem vorliegenden Diagonalscrambler für den Untercode $\mathcal{B}^{(2)}$ die gleiche freie Distanz

$$d^{(2)} = 6.$$

wie bereits mit dem Blockscrambler aus Beispiel 3.4. Im zweiten Partitionierungsschritt kann die freie Distanz jedoch im Gegensatz zum Blockscrambler weiter erhöht werden, man erhält

$$d^{(3)} = 8,$$

da alle Pfade vom Gewicht 7 punktiert werden, Pfad 8.3 mit Gewicht 8 jedoch nicht.

Wie man sieht, kann man mit einem Diagonalscrambler offenbar eine größere Distanzerhöhung erzielen als mit einem Blockscrambler.

□

Unit Memory Scrambler in allgemeiner Form

Eine weitere Verallgemeinerung bezüglich Gleichung 3.76 kann man erzielen, wenn man die durch Gleichung 3.66 vorgegebene Diagonalstruktur aufhebt und beliebige Matrizen $\underline{P}_0$ und $\underline{P}_1$ zuläßt. Bestimmte Einschränkungen sind jedoch auch dann notwendig. Sie werden nun zunächst kurz aufgeführt, anschließend werden ihre Auswirkungen auf die Scramblerkonstruktion betrachtet:

1. Keines der K Informationsbits $\underline{z}_{\tau-1}$, die einem Übergang Γ_τ im UM-Trellis zugeordnet werden, darf ausschließlich vom alten Zustand $\Theta_{\tau-1}$ abhängen.
2. Damit die lineare Abbildung eindeutig umkehrbar ist, muß die Matrix $\underline{M}(D)$ existieren, das heißt die inverse Scramblermatrix $M(D)^{-1}$ muß invertierbar sein. Dies ist nur der Fall, wenn $\det(M(D)^{-1} \neq 0$ gilt.
3. Durch $M(D)$ muß ein realisierbarer Code bzw. Scrambler beschrieben werden. Das heißt, die Determinante $\det\left(\underline{M}(D)^{-1}\right)$ der inversen Scramblermatrix muß die Form $1 + \dots$ haben.
4. Damit die Decodierkomplexität nicht erhöht wird und eine Decodierung im konventionellen Trellis durch einfaches Mapping der Übergänge möglich ist, darf kein Informationsbit z^i_τ von mehr als $\nu + 1$ aufeinanderfolgenden Bits x^i_τ abhängen.

Zu Punkt 1:

Diese Einschränkung ist gleichbedeutend mit dem zweiten und dritten Punkt, da bei einer Verletzung dieser Einschränkung die betreffende Spalte der inversen Scramblermatrix $\underline{M}(D)^{-1}$ nur Elemente 0 oder D enthält. Dies hat jedoch $\det(\underline{M}(D)^{-1}) = 0$ oder $\det(\underline{M}(D)^{-1}) = D \cdot (\dots)$ zur Folge.

Zu Punkt 2:

Diese Einschränkung bedeutet, daß die Spalten der inversen Scramblermatrix linear unabhängig sein müssen. Diese Einschränkung gilt bereits für Block- und Diagonalscrambler und läßt sich leicht realisieren, wenn die Konstruktion von M^{-1} schrittweise Spalte für Spalte erfolgt, wie in den obigen Beispielen.

Zu Punkt 3:

Wählt man wie beim Block- oder Diagonalscrambler die ersten $K-1$ Spalten im Hinblick auf die Punktierung bestimmter Zustände oder Übergänge im Trellisdiagramm, so kann diese Bedingung $\det(M(D)^{-1}) = 1 + \dots$ meist mit Hilfe der letzten Spalte des inversen Scramblers erfüllt werden, da diese prinzipiell frei wählbar ist.
Falls möglich sollte außerdem $\det(M(D)^{-1}) = 1$ erzielt werden, der Scrambler stellt dann einen nichtrekursiven Code dar. Im Rahmen dieser Arbeit wurden auch rekursive Scrambler konstruiert, da jedoch in allen Fällen gleichwertige nichtrekursive Scrambler existierten, wurden nur diese näher betrachtet. Insbesondere wurden sämtliche Simulationen nur mit nichtrekursiven Scramblern durchgeführt.

Zu Punkt 4:

Dieser Punkt stellt die größte, gleichzeitig aber auch die wichtigste Einschränkung dar.

Prinzipiell kann auch ein Scrambler konstruiert und eingesetzt werden, der diese Bedingung nicht erfüllt, doch muß dann die Decodierung im Trellis des äquivalenten Codes $G''(D) = \underline{M}(D) \cdot \underline{G}'(D)$ erfolgen und ist nicht mehr im konventionellen Trellis möglich. Da dessen Gedächtnislänge im allgemeinen Fall größer als die des Ausgangscodes $G(D)$ ist,

ist auch die Zweigkomplexität (bzw. die Anzahl der Zustände und Übergänge) größer als die des konventionellen Trellis. Die Decodierkomplexität steigt damit an.
Wird obige Beschränkung jedoch eingehalten, so kann die Decodierung ohne wesentliche Erhöhung der Komplexität im konventionellen Trellis erfolgen. Hierzu ist lediglich ein zusätzliches Mapping der Übergänge (oder Zustände) nötig, welches über eine einfache Tabelle realisiert werden kann (siehe hierzu Kapitel 4.2).

Unter Berücksichtigung dieser Überlegungen kann die Matrix $\underline{P}$ prinzipiell folgende Form annehmen: Die Länge l der Vektoren $\underline{p}^{(i)}$ muß dabei nach Punkt 4 kleiner gleich $\nu+1$ gewählt

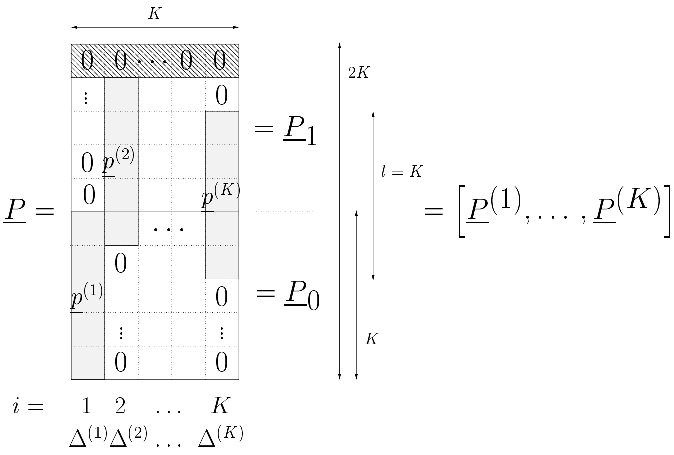


Abbildung 3.11: Teilmatrizen eines inversen UM-Scramblers

werden. Für die bisher betrachteten Fälle $K = \nu$ sowie $K = \nu + 1$ ist dies gewährleistet, wenn man wie bei Block- und Diagonalscramblern $l = K$ wählt. Davon soll nun zunächst ausgegangen werden. Der allgemeinere Fall $l \neq K$ wird im nächsten Abschnitt betrachtet.

Die Verschiebung $\Delta^{(i)}$ des Vektors $\underline{p}^{(i)}$ in der i-ten Spalte $\underline{P}^{(i)}$ ist nun aber nicht mehr starr vorgegeben wie beim Diagonalscrambler, sondern frei wählbar. Unter Berücksichtigung von Punkt 1 ergibt sich jedoch folgende Einschränkung:

$$0 \leq \Delta^{(i)} \leq K - 1 \tag{3.77}$$

Sie stimmt überein mit der Definition der Verschiebung Δ auf Seite 25 und den Überlegungen auf Seite 38. In Bild 3.11 ist diese Einschränkung durch die schraffierte oberste Zeile markiert. Sie kann nach Gleichung 3.77 nie von einem der Vektoren $\underline{p}^{(i)}$ belegt sein, ihre Elemente sind immer Null.

Betrachtet man die Wirkung des so auf theoretischem Wege verallgemeinerten UM-Scramblers im Trellis, so gilt folgendes: Die Bits der Partitionierungsfolge $\underline{z}^{(i)}$ hängen (wie auch bei einem BS oder DS) ausschließlich von der i-ten Spalte $\underline{P}^{(i)}$ der inversen Scramblermatrix ab:

$$z_{\tau-1}^{(i)} = \underline{\Gamma}_\tau \cdot \underline{P}^{(i)}; \tag{3.78}$$

Dies ist in Bild 3.12 dargestellt. Abhängig von der Wahl der Partitionierungsart bzw. der Scramblerdimension K, kann man diesen Zusammenhang analog zu den Gleichungen 3.57 und 3.60 mit den Elementen des konventionellen Trellis beschreiben (siehe 3.81 und 3.84, Seite 46).

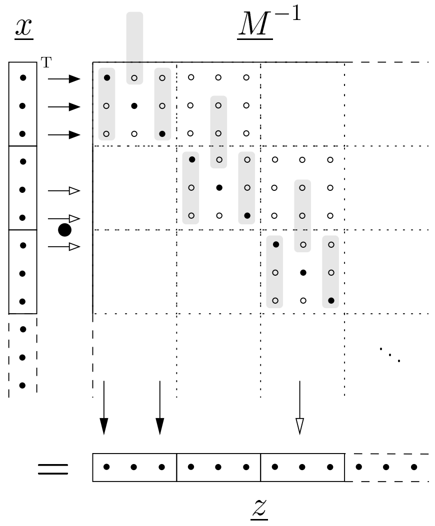


Abbildung 3.12: Abbildung von $\underline{z}$ auf $\underline{x}$ durch einen inversen UM-Scrambler

Im Unterschied zum Diagonalscrambler, muß man zur eindeutigen Beschreibung eines inversen UM-Scramblers nicht nur die Partitionierungsvektoren $\underline{p}^{(1)}, \ldots, \underline{p}^{(K)}$ angeben, sondern zusätzlich die Verschiebungen $\Delta^{(1)}, \ldots, \Delta^{(K)}$. Neben den bisher beschriebenen Darstellungen $\underline{P}$ und $\underline{M}(D)^{-1}$ kann man daher eine neue Form der Darstellung definieren, die (p, Δ)-Darstellung:

$$\left[\left(\underline{p}^{(1)}, \Delta^{(1)}\right), \ldots, \left(\underline{p}^{(K)}, \Delta^{(K)}\right)\right] \quad \Longleftrightarrow \quad \underline{P} \tag{3.79}$$

Der Zusammenhang zwischen einer Spalte $\underline{P}^{(i)}$ des inversen Scramblers und ihrer (p, Δ)-Darstellung lautet:

$$\underline{P}^{(i)} = \underline{H}^{\Delta^{(i)}} \cdot \underline{p}^{(i)}, \qquad \text{mit} \quad \underline{H} = \begin{bmatrix} 0 & \mathbf{1} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \mathbf{1} & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ & & & & \mathbf{1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \mathbf{1} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{3.80}$$

Bei der Konstruktion eines inversen Unit Memory Scramblers muß demnach für jede Spalte nicht nur ein Vektor $\underline{p}^{(i)}$, sondern eine Kombination $(p^{(i)}, \Delta^{(i)})$ bestimmt werden, durch die möglichst viele Pfade vom Gewicht $d^{(i)}$ punktiert werden. Die Menge $\mathbf{P}^{(i)}$ von Partitionierungsvektoren nach Gleichung 3.68 oder 3.71 geht daher über in eine Menge $\mathbf{P}_{\Delta}^{(i,)}$, die auch von der Wahl der Verschiebung Δ abhängt (siehe 3.82 und 3.85).
Diese Abhängigkeit kommt von der Menge $\overline{\mathbf{X}}_{\Delta^{(i)}}^{(i)}$, die i.a. für unterschiedliche Verschiebungen verschieden gewählt werden muß. Prinzipiell muß man die Menge $\mathbf{P}_{\Delta}^{(i)}$ ermitteln, deren $(p^{(i)}, \Delta)$-Kombinationen die größte Distanzerhöhung im linearen Untercode erzielen (Siehe hierzu auch Abschnitt 3.6.3).

Auch die Berechnung der Mengen $\overline{\mathbf{V}}_{\Delta^{(i)}}^{(i)}$ ändert sich geringfügig. Man muß berücksichtigen, daß möglicherweise bereits durch eine frühere Partitionierungsstufe Übergänge (oder

Zustände) zu den Zeitpunkten $t = K\tau - \Delta^{(i)}$ punktiert wurden, da sich zwei Stufen auf die gleichen Zeitpunkte beziehen können ($\Delta^{(i)} = \Delta^{(j)},\ i \neq j$). Dieser Fall tritt zum Beispiel bei jedem Blockscrambler auf. Man berücksichtigt dies bei der Berechnung, indem man die Vereinigungsmenge aus den bereits punktierten und den neu punktierten Übergängen (oder Zuständen) berechnet (siehe Gleichungen 3.83 und 3.86).

Die wichtigsten Gleichungen zur Beschreibung eines inversen UM-Scramblers sind im folgenden für beide möglichen Punktierungsmethoden zusammengefaßt. Sie stellen die Verallgemeinerung der Gleichungen 3.67 bis 3.72 dar.

- **Zustandspunktierung** ($K = \nu$) mit UM-Scrambler:

$$z_{\tau-1}^{(i)} = \underline{\theta}_t \cdot \underline{p}^{(i)} \qquad \text{für} \quad t = K \cdot \tau - \Delta^{(i)} \tag{3.81}$$

$$\mathbf{P}_{\Delta}^{(i)} = \left\{ \ \underline{p}^{(i)} \ \middle| \ \underline{\theta}_t \cdot \underline{p}^{(i)} = 1 \qquad \forall\, \underline{\theta}_t \in \overline{\mathbf{X}}_{\Delta}^{(i)} \ \right\} \tag{3.82}$$

$$\overline{\mathbf{V}}_{\Delta^{(i)}}^{(i)} = \overline{\mathbf{V}}_{\Delta^{(i)}}^{(i-1)} \cup \left\{ \underline{\theta}_t \in \mathbf{V} \,\middle|\, \underline{\theta}_t \cdot \underline{p}^{(i)} = 1 \right\} \tag{3.83}$$

- **Übergangspunktierung** ($K = \nu + 1$) mit UM-Scrambler:

$$z_{\tau-1}^{(i)} = \underline{\gamma}_t \cdot \underline{p}^{(i)} \qquad \text{für} \quad t = K \cdot \tau - \Delta^{(i)} \tag{3.84}$$

$$\mathbf{P}_{\Delta}^{(i,)} = \left\{ \ \underline{p}^{(i)} \ \middle| \ \underline{\gamma}_t \cdot \underline{p}^{(i)} = 1 \qquad \forall\, \underline{\gamma}_t \in \overline{\mathbf{X}}_{\Delta}^{(i)} \ \right\} \tag{3.85}$$

$$\overline{\mathbf{V}}_{\Delta^{(i)}}^{(i)} = \overline{\mathbf{V}}_{\Delta^{(i)}}^{(i-1)} \cup \left\{ \underline{\gamma}_t \in \mathbf{V} \,\middle|\, \underline{\gamma}_t \cdot \underline{p}^{(i)} = 1 \right\} \tag{3.86}$$

Beispiel 3.6:

Für den Code $(5,7)$ der in Beispiel 3.5 mit einem Diagonalscrambler 3-fach partitioniert wurde, soll jetzt ein UM-Scrambler konstruiert werden. Hierzu wird noch einmal Tabelle 3.5 betrachtet.

Das Ergebnis des ersten Partitionierungsschritts ist unabhängig von der Verschiebung Δ (siehe Kapitel 3.6.1), daher wird hier willkürlich $\Delta^{(1)} = 0$ gewählt. Um alle Codefolgen vom Gewicht $d^{(1)} = 5$ zu punktieren, müssen die Übergänge 1,2 und 4 punktiert werden. Wendet man obige Gleichungen an, so ergibt sich:

$$\overline{\mathbf{X}}_0^1 = \{1,2,4\} \quad \Rightarrow \quad \mathbf{P}_0^{(1)} = \{(111)^T\}$$

$$\Rightarrow \quad \underline{p}^{(1)} = (111)^T \quad \Rightarrow \quad \overline{\mathbf{V}}_0^{(1)} = \{1,2,4,7\}$$

Die freie Distanz kann also wie gewünscht erhöht werden. Da noch Pfade vom Gewicht 6 existieren gilt $d^{(2)} = 6$.

Im nächsten Partitionierungsschritt muß man nun für jede Verschiebung Δ eine geeignete Punktierungsmenge $\overline{\mathbf{X}}_{\Delta}^{(i)}$ bestimmen, um eine weitere Distanzerhöhung zu erzielen. Nach Tabelle 3.5 ergeben sich folgende Möglichkeiten:

$$
\begin{array}{llll}
\Delta^{(2)} = 0 & \overline{\mathbf{X}}_0^2 = \{3,5,6\} & \Rightarrow & \mathbf{P}_0^{(2)} = \{\ \} \\
\Delta^{(2)} = 1 & \overline{\mathbf{X}}_1^2 = \{1,2,3\} & \Rightarrow & \mathbf{P}_1^{(2)} = \{\ \} \\
 & \overline{\mathbf{X}}_1^2 = \{3,4\} & \Rightarrow & \mathbf{P}_1^{(2)} = \{\ \} \\
\Delta^{(2)} = 2 & \overline{\mathbf{X}}_2^2 = \{1,6\} & \Rightarrow & \mathbf{P}_2^{(2)} = \{(101)^T,(011)^T\} \\
 & \overline{\mathbf{X}}_2^2 = \{2,6\} & \Rightarrow & \mathbf{P}_2^{(2)} = \{(010)^T,(011)^T\}
\end{array}
$$

Man sieht folgendes: Es existieren keine Vektoren $\underline{p}^{(2)}$, mit denen man die gewünschte Punktierung zu Zeitpunkten mit der Verschiebung 0 oder 1 realisieren könnte. Dagegen gibt es 3 Vektoren, die eine Distanzerhöhung durch Punktierung für $\Delta^{(2)} = 2$ ermöglichen. Sie punktieren die folgenden Übergänge:

$$
\begin{array}{llll}
a) & \underline{p}_a^{(2)} = (101)^T & \Rightarrow & \overline{\mathbf{V}}_2^{(2)} = \{\ \} \cup \{1,3,4,6\} \\
b) & \underline{p}_b^{(2)} = (011)^T & \Rightarrow & \overline{\mathbf{V}}_2^{(2)} = \{\ \} \cup \{1,2,5,6\} \\
c) & \underline{p}_c^{(2)} = (010)^T & \Rightarrow & \overline{\mathbf{V}}_2^{(2)} = \{\ \} \cup \{2,3,6,7\}
\end{array}
$$

Betrachtet man Tabelle 3.5 und berücksichtigt die bereits durch $\underline{p}^{(1)}$ punktierten Pfade, so stellt man fest, daß der Vektor $\underline{p}_b^{(2)}$ als einziger neben den Pfaden mit Gewicht 6 auch alle Pfade vom Gewicht 7 punktiert. Man wählt daher für die zweite Spalte der inversen Scramblermatrix $\Delta^{(2)} = 2$ und $\underline{p}^{(2)} = (011)^T$.

Durch geeignete Wahl der letzten Spalte erhält man dann

$$
\underline{P} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 \\ 0 & \mathbf{1} & 0 \\ & & \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & 0 & \mathbf{0} \\ \mathbf{1} & 0 & \mathbf{1} \end{bmatrix} \qquad \text{bzw.} \qquad \underline{M}(D)^{-1} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & D & 1 \end{bmatrix}
$$

Die freien Distanzen der linearen Untercodes, die man mit Hilfe dieses UM-Scramblers erzielt lauten:

$$
d^{(2)} = 6, \qquad d^{(3)} = 8
$$

Vergleicht man sie mit dem Ergebnis in Beispiel 3.5, so stellt man fest, daß für den Code $(5,7)$ bei dreifacher Partitionierung durch den UM-Scrambler offenbar keine größere Distanzerhöhung erzielt werden kann, als mit dem Diagonalscrambler. Die Begründung hierfür findet man in Kapitel 3.6.1 (Beispiel 3.7): Die beiden Scrambler sind „partitionierungsäquivalent“.

Im allgemeinen Fall erzielt man mit UM-Scramblern jedoch bessere Ergebnisse, als mit Diagonalscramblern (siehe Anhang A.1).

□

3.5.3 Faltungsscrambler (CS)

Prinzipiell kann man die zuvor beschriebene Partitionierung auch dann unverändert anwenden, wenn man mehr oder weniger als ν bzw. $\nu + 1$ Partitionierungsstufen benötigt. Das heißt man kann eine weitere Verallgemeinerung vornehmen, indem man den Parameter der Scramblerdimension K von der Partitionierungsart (Zustandspunktierung oder Übergangspunktierung) und somit von der Länge l der Vektoren $\underline{p}^{(i)}$ trennt.

Die Partitionierung durch Zustandspunktierung ($l = \nu$) kann dann als Spezialfall der Partitionierung durch Übergangspunktierung ($l = \nu + 1$) angesehen werden, bei dem das „oberste“ Bit $p_1^{(i)}$ aller Vektoren $p^{(i)}$ identisch 0 ist. Die Partitionierung durch Übergangspunktierung besitzt also einen Freiheitsgrad mehr, wird im allgemeinen bessere Ergebnisse liefern und sollte daher der Zustandspunktierung vorgezogen werden. Man sollte also $l = \nu + 1$ wählen.
Die Anzahl der Partitionierungsstufen, also die Scramblerdimension K kann prinzipiell frei gewählt werden.

Wählt man $K < l$, so kann dies zur Folge haben, daß die Gedächtnislänge des inversen Scramblers größer als 1 wird. Diese Art von Scrambler soll als *Faltungsscrambler* (Convolutional Scrambler, CS) bezeichnet werden. Die Menge der Faltungsscrambler enthält die Menge der UM-Scrambler ebenso wie die Menge der Blockscrambler.

In Bild 3.13 auf Seite 49 sind alle möglichen Scrambler symbolisch in einer Tabelle dargestellt. In horizontaler Richtung ist dabei die Scramblerdimension K aufgetragen, in vertikaler Richtung die Punktierungsart und die Scramblerart.
Die schraffierten Bereiche in der Tabelle kennzeichnen die Scramblerarten, welche in [BDS96a], [BDS96b] und [Die96] beschrieben und realisiert sind. In der vorliegenden Arbeit werden sie in den Abschnitten 3.5.1 und 3.5.2 behandelt.
Die allgemeinen UM-Scrambler nach Abschnitt 3.5.2 sowie allgemeine Faltungsscrambler stellen eine Weiterentwicklung dar und wurden erstmals im Rahmen der vorliegenden Arbeit realisiert. Erste Ideen hierzu findet man bereits in [Die96].
Für $K < l$ existieren im allgemeinen keine Blockscrambler, wenn man die partitionierenden Vektoren $\underline{p}^{(i)}$ in voller Länge l nutzt.

Für $K > l$ existieren wegen der Forderung $\det(\underline{M}(D)^{-1}) \neq 0$ strenggenommen keine inversen Blockscrambler im Sinne von Kapitel 3.5.1 für die sowohl $m_{\text{inv}} = 0$ als auch $\Delta^{(i)} = 0$ für alle Spalten i gilt. Es existieren jedoch inverse Scrambler, die die erste Bedingung $m_{\text{inv}} = 0$ erfüllen. Auch diese werden in Analogie zu Blockcodes als (inverse) Blockscrambler bezeichnet (siehe Bild 3.13).

In Tabelle 3.6 sind abschließend alle verwendeten Unterscheidungskriterien für (inverse) Scrambler aufgeführt.

Unterscheidungskriterium:	Möglichkeiten:		
Anzahl der Partitionierungsstufen = Dimension des Scramblers K	$\ldots, \nu, \nu+1, \ldots$		
Länge l der Partitionierungsvektoren Partitionierungsart Abkürzung	ν Zustände (states) s	$\nu+1$ Übergänge (transitions) t	
Gedächtnis m_{inv} Scramblerart Abkürzung	0 Blockscrambler BS	1 UM-Scrambler UMS	≥ 2 Faltungsscrambler CS
Verschiebung $\Delta^{(i)}$ der Part.vektoren Struktur, Abkürzung	0 Block BS	$K-i$ diagonal DS (UMS,CS)	$0 \leq \Delta^{(i)} \leq K-1$ beliebig UMS,CS

Tabelle 3.6: Unterscheidungskriterien von Faltungsscramblern

3.6 Konstruktion von Faltungsscramblern

Bevor nun im Abschnitt 3.6.3 ein Algorithmus zur Konstruktion bzw. Suche von inversen Faltungsscramblern angegeben wird, soll in Abschnitt 3.6.1 ein Effekt betrachtet werden, der diese Suche vereinfacht. Außerdem wird in Abschnitt 3.6.2 dargestellt, wie sich die Punktierung eines Faltungscodes auf die Scramblerkonstruktion auswirkt.

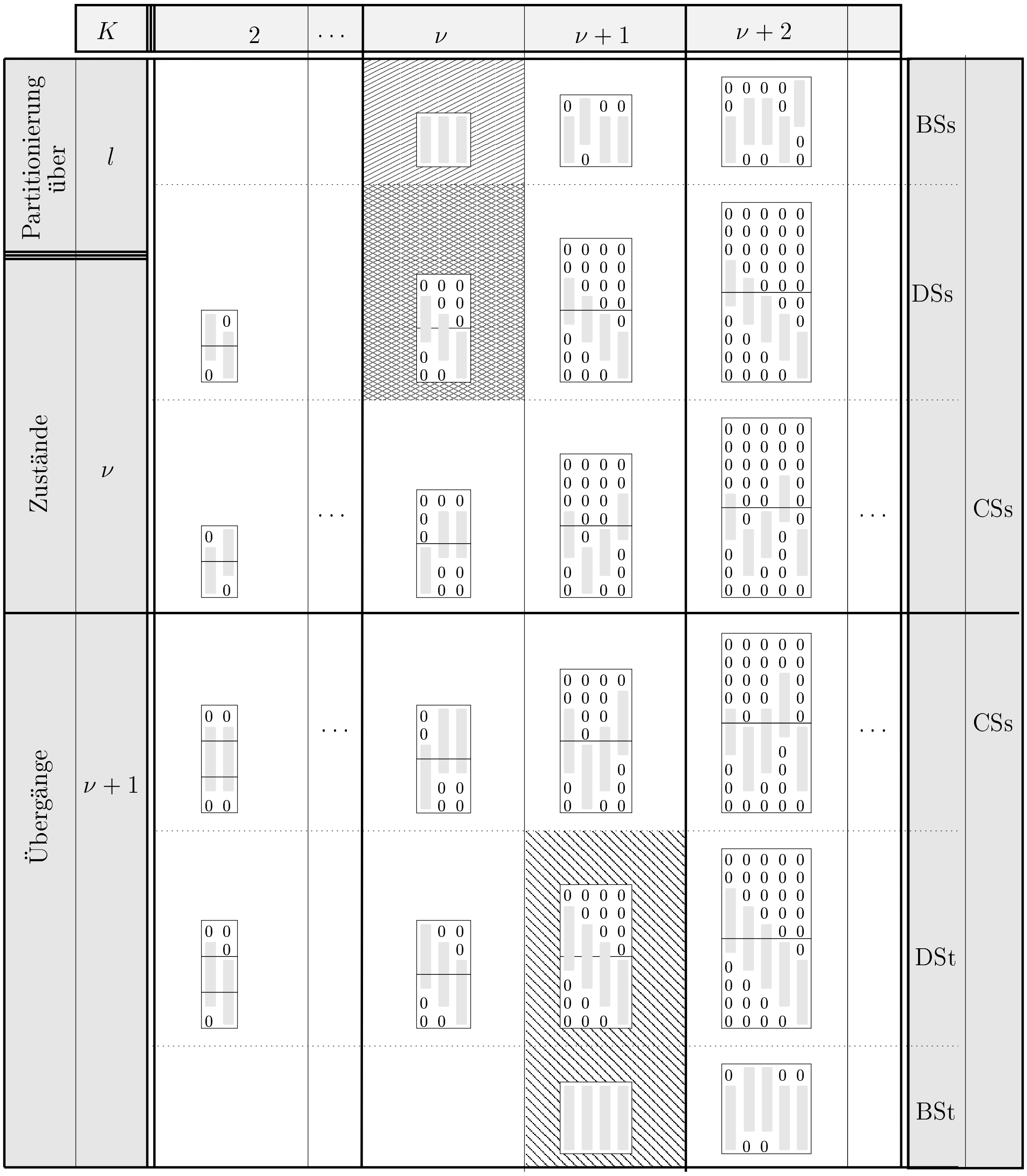


Abbildung 3.13: Verschiedene inverse Scrambler (am Beispiel $\nu = 3$)

3.6.1 Partitionierungsäquivalenz von Scramblern

Wie aus Tabelle A.3 ersichtlich, kann es vorkommen, daß mit verschiedenen Scrambler die gleichen freien Distanzen für die (linearen) Untercodes erzielt werden können. Bewirken zwei Scrambler darüberhinaus, daß die Menge von Codesequenzen mit dem Hamming-Gewicht d in den erzeugten linearen Untercodes für beide Scrambler und beliebiges Hamming-Gewicht genau gleich groß ist, so sollen diese Scrambler als *weitläufig partitionierungsäquivalent* bezeichnet werden.
Die Partitionierungsäquivalenz von Scramblern ist interessant, da sie bei genauerer Betrachtung die Scramblersuche vereinfacht. Hat man einen guten Scrambler gefunden, so kann man von ihm direkt weitere Scrambler ableiten, die ab sofort als *partitionierungsäquivalent* bezeichnet werden. Weshalb und wie das möglich ist, wird nun anhand des Codes $(5,7)$ erläutert und im Suchalgorithmus in Abschnitt 3.6.3 angewandt.

Betrachtet man den Code $(5,7)$, so existiert genau eine Codesequenz mit dem Hamming-Gewicht 5 und zwei Sequenzen mit dem Hamming-Gewicht 6. Betrachtet man dagegen zur K-fachen Partitionierung die UM-Darstellung des gleichen Codes, so muß man, wie in den Tabellen 3.2, 3.4 und 3.5 dargestellt, auch die um $1,\ldots,K-1$ Bit verschobenen Sequenzen berücksichtigen. Da der ursprüngliche Faltungscode zeitinvariant ist, besitzen diese Folgen natürlich das gleiche Hamming-Gewicht. Im Fall des Codes $(5,7)$ erhält man so insgesamt 3 Codefolgen mit dem Gewicht 5 und 6 vom Gewicht 6.

Bei der Partitionierung (mit beliebigen Faltungsscramblern) wirkt sich dies folgendermaßen aus:
Im ersten Partitionierungsschritt spielt es keine Rolle, zu welchem der Zeitpunkte $\Delta = 0,\ldots,K-1$ man die Punktierung bestimmter Übergänge (oder Zustände) vornimmt, da der Punktierungsvektor $\underline{p}^{(i)}$ unabhängig davon immer die gleiche Anzahl von Codefolgen eines bestimmten Gewichts punktiert. Nur dies ist entscheidend und es spielt keine Rolle, ob dabei im Beispiel nach Tabelle 3.5 der Pfad 1.1, 1.2 oder 1.3 punktiert wird.
Erst in den weiteren Partitionierungsschritten ist auch die Wahl von $\Delta^{(i)}$ in Kombination mit der Wahl des Vektors $\underline{p}^{(i)}$ entscheidend für die erzielte Distanzerhöhung, da nun nicht mehr mit beliebiger Verschiebung Δ jeder Übergang eines Pfades punktiert werden kann.

Da der Absolutwert der ersten Verschiebung $\Delta^{(1)}$ unerheblich für die „Qualität“ des Scramblers ist, solange die Differenzen zwischen den einzelnen Verschiebungen gleich bleiben, kann man die Partitionierungsäquivalenz mathematisch mit Hilfe der Modulo-Rechnung beschreiben (siehe [Haa96]). Alle Scrambler, deren (p,Δ)-Darstellung sich schreiben läßt als

$$\left[\left(\underline{p}^{(1)}, \Delta^{(1)} + \delta \bmod K\right), \ldots, \left(\underline{p}^{(K)}, \Delta^{(K)} + \delta \bmod K\right)\right] \tag{3.87}$$

sind partitionierungsäquivalent zum Scrambler nach Gleichung 3.79. Dabei stellt die Addition von δ eine zusätzliche Verschiebung aller Vektoren um δ in der Matrix $\underline{P}$ aus Gleichung 3.76 dar. Die Modulo-Rechnung bewirkt dabei anschaulich, daß der Teil eines Vektors, der oben aus $\underline{P}_1$ herausgeschoben wird, wieder unten in $\underline{P}_0$ auftaucht (siehe Bild 3.14).

Für UM-Scrambler läßt sich dieser Zusammenhang auch mit Hilfe einer Matrixmultiplikation ausdrücken, die man direkt auf die polynomiale inverse Scramblermatrix anwenden kann.

Mit Hilfe der Matrix

$$\underline{H}(D) = \begin{bmatrix} 0 & \mathbf{1} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \mathbf{1} & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ & & & & \mathbf{1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \mathbf{1} \\ \mathbf{D} & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{3.88}$$

erhält man K partitionierungsäquivalente inversen Scramblermatrizen:

$$\underline{H}(D)^{\delta} \cdot \underline{M}(D)^{-1}, \quad 0 \leq \delta \leq K-1 \tag{3.89}$$

Dabei ist noch folgendes zu beachten: Ist in einer dieser Matrizen das oberste Element einer Spalte gleich D, so müssen noch alle Elemente dieser Spalte durch D geteilt werden. Damit wird berücksichtigt, daß $\Delta < K$ gelten muß (siehe Beispiel 3.7).

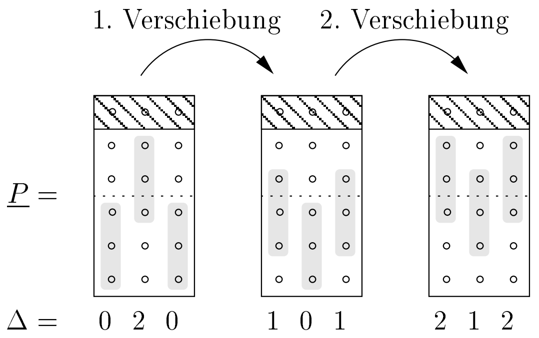


Abbildung 3.14: Graphische Darstellung der Partitionierungsäquivalenz

Die (p, Δ)-Darstellung für bestimmte Spalten bzw. Matrizen ist nicht eindeutig, daher können insgesamt noch mehr partitionierungsäquivalente Scramblermatrizen existieren.

Da die Determinante $\det(\underline{M}^{-1})$ nicht für alle partitionierungsäquivalenten inversen Scrambler identisch sein muß, kann es sein, daß die letzte Spalte neu gewählt werden muß, um einen realisierbaren Scrambler zu erhalten. Da die letzte Spalte aber keine Auswirkung auf die Partitionierungseigenschaften eines Scramblers hat, ist dies zulässig. Außerdem können die partitionierungsäquivalenten Scrambler prinzipiell rekursiv sein, auch wenn man von einem nichtrekursiven Scrambler ($\det(\underline{M}(D)^{-1}) = 1$) ausgegangen ist.

Beispiel 3.7:

Ausgehend von Beispiel 3.6 ergeben sich die folgenden 3 zueinander partitionierungsäquivalenten inversen Scrambler:

$$\underline{M}_0(D)^{-1} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & D & 1 \end{bmatrix}, \quad \underline{M}_1(D)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & D & 1 \\ D & D & 0 \end{bmatrix}, \quad \underline{M}_2(D)^{-1} = \begin{bmatrix} 1 & 1 & 1 \\ D & 1 & 0 \\ D & 0 & 0 \end{bmatrix}$$

Über verschiedene (p, Δ)-Darstellungen erhält auch noch den Scrambler

$$\underline{M}_{1b}(D)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ D & 1 & 0 \end{bmatrix}.$$

Die gefundenen Matrizen besitzen die Determinanten

$$\det(\underline{M}_0(D)^{-1}) = 1, \qquad \det(\underline{M}_1(D)^{-1}) = D^2 + D, \qquad \det(\underline{M}_2(D)^{-1}) = D,$$

$$\det(\underline{M}_{1b}(D)^{-1}) = 1,$$

Das heißt für die Fälle 1 und 2 muß die letzte Spalte neu gewählt werden, da die Scrambler sonst nicht realisierbar sind.

Bei $\underline{M}_0(D)^{-1}$ und $\underline{M}_2(D)^{-1}$ handelt es sich um den inversen UM-Scrambler aus Beispiel 3.6 und den inversen Diagonalscrambler aus Beispiel 3.5 (abgesehen von der letzten Spalte). Hiermit ist nun klar, warum für den betrachteten Fall mit dem Diagonalscrambler die gleiche Distanzerhöhung erreicht wird wie mit dem besten UM-Scrambler: beide sind partitionierungsäquivalent.

□

In den meisten Fällen handelt es sich bei den partitionierungsäquivalenten Scramblern eines UM-Scramblers nicht um Diagonalscrambler. In Kapitel 5 und in Anhang A.1 sind solche Fälle aufgeführt. Man sieht, daß mit einem UM-Scrambler im allgemeinen höhere freie Distanzen erzielt werden.

3.6.2 Besonderheiten bei punktierten Codes

Die Partitionierung von punktierten Faltungscodes kann prinzipiell genauso erfolgen, wie oben beschrieben, wenn man folgendes beachtet:
Beschreibt man einen nicht punktierten Faltungscodes zur Partitionierung als UM-Code, so existieren im UM-Trellis jeweils K Pfade mit gleichem Hamming-Gewicht, die sich im Trellis des konventionellen Codes nur durch die Verschiebung $\Delta = 0, \ldots, K-1$ des Zeitpunktes t unterschieden, in dem sie vom Nullzustand abweichen (siehe Abschnitt 3.6.1 und Tabelle 3.7 und 3.8).

Bei punktierten Codes besitzen die so verschobenen Pfade aufgrund der immer gleichbleibenden Punktierung unter Umständen unterschiedliche Hamming-Gewichte, da nur die übertragenen Codebits zum Gewicht beitragen.

Gilt für die Punktierungsperiode nicht $p_{\text{per}} = K$, so können außerdem auch Pfade, die zu verschiedenen Zeitpunkten mit gleicher Verschiebung Δ starten, aufgrund unterschiedlicher Punktierung verschiedene Hamming-Gewichte besitzen. Dies ist für den nach Beispiel 2.9 auf Rate $r = 4/5$ punktierten Codes $(5, 7)$ für 2-stufige Partitionierung $(K = 2)$ in Tabelle 3.7 und für 3-stufige Partitionierung $(K = 3)$ in Tabelle 3.8 dargestellt.

Im allgemeinen muß man für die Partitionierung eines punktierten Faltungscodes nicht nur K Verschiebungen, sondern insgesamt

$$\text{kgV}\,(K, p_{\text{pkt}}) \tag{3.90}$$

Verschiebungen derselben Codefolge des unpunktierten Codes berücksichtigen. Mit „kgV“ ist dabei das kleinste gemeinsame Vielfache der beiden Werte gemeint.
Von diesen Pfaden werden durch Zustands- oder Übergangspunktierung immer mehrere gleichzeitig entfernt, die sich nur in der Punktierung (und im Gewicht) unterscheiden. In Kapitel 5 werden die Ergebnisse der Partitionierung eines punktierten Faltungscodes aufgeführt.

Nr.	Pfad $\underline{\theta}$ mit den Zuständen $\underline{\theta}_t$ fett: UM-Zustände $\underline{\Theta}_\tau$				Gewicht
1.1.1	**1** 2				4
1.2.1	**0** 1	**2**			2
1.1.2	**0** 0	**1** 2			3
1.2.2	**0** 0	**0** 1	**2**		3

Tabelle 3.7: Codesequenzen des punktierten Codes $(5,7;r=4/5)$ als Zustandsfolgen

Nr.	Pfad $\underline{\gamma}$ mit den Übergängen $\underline{\gamma}_t$ fett: UM-Zustände $\underline{\Theta}_\tau$					Gewicht
1.1.1	**1** 2 4					4
1.2.1	**0** 1 2	**4**				2
1.3.1	**0** 0 1	**2** 4				3
1.1.2	**0** 0 0	**1** 2 4				3
1.2.2	**0** 0 0	**0** 1 2	**4**			4
1.3.2	**0** 0 0	**0** 0 1	**2** 4			2
1.1.3	**0** 0 0	**0** 0 0	**1** 2 4			3
1.2.3	**0** 0 0	**0** 0 0	**0** 1 2	**4**		3
1.3.3	**0** 0 0	**0** 0 0	**0** 0 1	**2** 4		4
1.1.4	**0** 0 0	**0** 0 0	**0** 0 0	**1** 2 4		2
1.2.4	**0** 0 0	**0** 0 0	**0** 0 0	**0** 1 2	**4**	3
1.3.4	**0** 0 0	**0** 0 0	**0** 0 0	**0** 0 1	**2** 4	3

Tabelle 3.8: Codesequenzen des punktierten Codes $(5,7;r=4/5)$ als Übergangsfolgen

3.6.3 Algorithmus zur Scramblerkonstruktion

Die Konstruktion eines inversen Scramblers $M(D)^{-1}$ für die Partitionierung eines Codes $\mathcal{B}^{(1)}$ stellt prinzipiell eine systematische Suche dar. Im Rahmen der vorliegenden Arbeit wurde ein Algorithmus zur Scramblersuche entwickelt und auf verschiedene Weisen implementiert. In Kapitel B.1.1 findet man die Beschreibung des Suchprogramms, das sich für größere Scramblerdimensionen als beste Lösung im Hinblick auf Speicher und Rechenzeit erwiesen hat.

In diesem Abschnitt soll nicht der gesamte, relativ komplexe Algorithmus exakt angegeben werden. Es wird nur der prinzipielle Ablauf der wichtigsten beiden Teile beschrieben.

Die Suche nach einem inversen Scrambler kann aufgeteilt werden in die Suche nach den einzelnen Matrixspalten $\underline{P}^{(i)}$ bzw. Kombinationen $(\underline{p}^{(i)}, \Delta^{(i)})$ in Abhängigkeit von einer bereits vorhandenen Teilmatrix $[\underline{P}^{(1)}, \ldots, \underline{P}^{(i-1)}]$.

Mit Hilfe der folgenden Definitionen

$l = \nu + 1$	zur Partitionierung durch Übergangspunktierung
Λ	$= \{0, 1, \ldots, K-1\}$, Menge aller zulässigen Verschiebungen Δ
$\mathbf{P}$	Menge aller möglichen 2^l binären Partitionierungsvektoren der Länge l
$\mathbf{K}$	$= \{(\underline{p}, \Delta) \mid \underline{p} \in \mathbf{P}, \Delta \in \Lambda\}$
$\mathbf{K}_1$	$= \{(\underline{p}, 1) \mid \underline{p} \in \mathbf{P}\}$
$\mathbf{K}_{\text{best}}$	Menge von $(\underline{p}, \Delta)$-Kombinationen
$\lvert\mathbf{K}_{\text{best}}\rvert$	Mächtigkeit der Menge $\mathbf{K}_{\text{best}}$
$\mathbf{E}_d^{(i)}$	Menge der Code-Übergangsfolgen $\underline{\gamma}$ des Untercodes $\mathcal{B}^{(i)}$ vom Gewicht d
$\underline{H}$	Verschiebematrix nach Gleichung 3.80
a_{grenz}	maximale Anzahl gleichwertiger Spalten, die der Algorithmus liefern soll
d_{grenz}	maximale Hamming-Distanz, die für die Suche berücksichtigt werden soll

lautet der Algorithmus zur Bestimmung einer Menge $\widehat{\mathbf{K}}^{(i)}$ geeigneter Matrixspalten $\underline{P}^{(i)}$:

1. Falls $(i = 1)$, dann setze $\mathbf{K}^{(i)} = \mathbf{K}_1$, sonst setze $\mathbf{K}^{(i)} = \mathbf{K}$.
2. Falls $i > 1$:
   Entferne alle Kombinationen aus $\mathbf{K}^{(i)}$, die Spalten $\underline{P}^{(i)}$ beschreiben, die linear abhängig von den bereits vorhandenen Matrixspalten $\underline{P}^{(1)}, \ldots, \underline{P}^{(i-1)}$ sind.
3. Setze $d = d^{(i)}$.
4. Bestimme die Menge $\mathbf{E}_d^{(i)}$ aller Übergangsfolgen $\underline{\gamma}$, die Codefolgen $\underline{y} \in \mathcal{B}^{(i)}$ vom Hamming-Gewicht d entsprechen.
5. Bestimme die Menge $\mathbf{K}_{\text{best}} \subset \mathbf{K}^{(i)}$ aller $(\underline{p}, \Delta)$-Kombinationen, welche durch Übergangspunktierung die größtmögliche Anzahl von Pfaden $\underline{\gamma}$ aus $\mathbf{E}_d^{(i)}$ punktiert.
6. Falls $(|\mathbf{K}_{\text{best}}| > a_{\text{grenz}})$ und $(d < d_{\text{grenz}})$:
   $d := d + 1$
   $\mathbf{K}^{(i)} := \mathbf{K}_{\text{best}}$
   Gehe zu Schritt 4.
7. Berechne zu allen Kombinationen $(\underline{p}^{(i)}, \Delta^{(i)}) \in \mathbf{K}^{(i)}$ den zugehörigen Vektor $\underline{P}^{(i)} = \underline{H}^{\Delta^{(i)}} \cdot \underline{p}^{(i)}$ und fasse sie zur Menge $\widehat{\mathbf{K}}^{(i)}$ zusammen.

Die so ermittelten Matrixspalten $\underline{P}^{(i)} \in \widehat{\mathbf{K}}^{(i)}$ sind gleichwertig bezüglich der Anzahl punktierter Pfade bis zum Gewicht d_{grenz} und nicht nur bezüglich der erzielten freien Distanz $d^{(i+1)}$. Da die Anzahl von Codefolgen mit wachsendem Gewicht d erheblich ansteigt, kann der Wert d_{grenz} im Hinblick auf den benötigten Speicher nicht beliebig groß gewählt werden. Das heißt es kommt vor, daß der Algorithmus mehrere Spalten $\underline{P}^{(i)}$ liefert, obwohl diese unterschiedlich viele Codefolgen vom Gewicht $d > d_{\text{grenz}}$ punktieren. Dies kann sich auf das Ergebnis der weiteren Partitionierungsschritte auswirken.

Man muß daher bei der weiteren Scramblerkonstruktion nicht nur die Punktierungsergebnisse verschiedener Spalten $\underline{P}^{(i+1)}$ vergleichen, sondern dabei auch die unterschiedlichen Teilmatrizen $[\underline{P}^{(1)}, \ldots, \underline{P}^{(i)}]$ berücksichtigen.

Dies wird im folgenden Algorithmus beschrieben, wobei zusätzlich folgende Bezeichnungen gelten:

$\widehat{\mathbf{K}}^{(i)}$	Menge von Matrixspalten $\underline{P}^{(i)}$
$\mathbf{M}^{(i)}$	Menge möglicher Teilmatrizen $[\underline{P}^{(1)}, \ldots, \underline{P}^{(i)}]$

1. i=1
2. Berechne die Menge $\widehat{\mathbf{K}}^{(1)}$ möglicher Matrixspalten $\underline{P}^{(1)}$ (s.o.) und setze die Teilmatrixmenge $\mathbf{M}^{(1)} := \widehat{\mathbf{K}}^{(1)}$.
3. i:=i+1

4. Berechne für jede der Teilmatrizen aus $\mathbf{M}^{(i-1)}$ die Menge $\widehat{\mathbf{K}}^{(i)}$ möglicher Matrixspalten $\underline{P}^{(i)}$ (s.o.) und vergleiche dabei die entstehenden Teilmatrizen untereinander bezüglich der Pfadpunktierung bis zum Gewicht d_{grenz}.
   Fasse von den so entstandenen Teilmatrizen $[\underline{P}^{(1)}, \ldots, \underline{P}^{(i)}]$ diejenigen zur Menge $\mathbf{M}^{(i)}$ zusammen, welche die beste Distanzverteilung für $\mathcal{B}^{(i+1)}$ erzielen (größte freie Distanz, wenig Pfade mit geringem Gewicht).

5. Falls gilt $(i \leq K-1)$, dann gehe zu Schritt 3

6. Bestimme folgendermaßen die Menge $\mathbf{M}^{(K)}$: Berechne für alle Teilmatrizen die letzte Spalte $\underline{P}^{(K)}$ so, daß sich $\det(\underline{M}(D)^{-1} = 1 + \ldots$ ergibt. Existieren mehrere Möglichkeiten, so wähle eine Spalte mit möglichst geringem Hamming-Gewicht. Existiert keine Möglichkeit, ist der (inverse) Scrambler nicht realisierbar und die Matrix $\underline{P}$ wird nicht in $\mathbf{M}^{(K)}$ aufgenommen.

7. Berechne für alle $\underline{P} \in \mathbf{M}^{(K)}$ die zugehörigen inversen Scrambler $\underline{M}(D)^{-1} = \underline{P}_0 + D \cdot P_1$.

Alle so ermittelten Scrambler sind gleichwertig bezüglich der Partitionierung des betrachteten Faltungscodes.

KAPITEL

4

Decodierung bei verallgemeinerter Verkettung

Um die Korrekturfähigkeiten eines verallgemeinert verketteten Codes (*GC-Codes*) zu nutzen genügt es nicht, die verwendeten Teilcodes getrennt zu decodieren. Stattdessen ist eine verschachtelte Decodierung von inneren und äußeren Codes notwendig. Diese wird in Abschnitt 4.1 beschrieben.
Zur Realisierung der Decodierung bei Verwendung von Faltungscodes benötigt man einen Decoder, der einerseits Zuverlässigkeitsinformationen verarbeiten kann und andererseits Wahrscheinlichkeitswerte für die Zuverlässigkeit der einzelnen Informationsbits ausgibt, also einen *Soft In - Soft Out* Decoder. In der vorliegenden Arbeit wurde hierfür der MAP-Algorithmus nach Bahl u.a. [BCJR74] ausgewählt, der jedoch in einigen Punkten modifiziert werden mußte. Die Änderungen sind in Abschnitt 4.2 beschrieben.

4.1 Decodierprinzip bei verallgemeinerter Codeverkettung

Das grundlegende Prinzip der Decodierung verallgemeinert verketteter Codes soll anhand von Bild 4.1 kurz erläutert werden. Es zeigt das Gesamtsystem, das in der vorliegenden Arbeit realisiert wurde. Für die Darstellung wurden drei Partitionierungsstufen gewählt, prinzipiell sind jedoch auch mehr oder weniger Stufen möglich.

Im oberen Abschnitt ist der Encoder dargestellt, der sich aus den Encodern der äußeren Codes $\mathcal{A}^{(i)}$, zwischengeschalteten Interleavern, dem Scrambler sowie dem Encoder des inneren Codes $\mathcal{B}^{(1)}$ zusammensetzt. Die Interleaver sind notwendig um die Fehlerfortpflanzung von einer Decodierungsstufe zur nächsten zu verhindern bzw. so gering wie möglich zu halten.
Im unteren Teil sieht man die verwendete Decoderstruktur. Um die volle Decodierfähigkeit des Gesamtcodes auszunutzen, findet eine verschachtelte Decodierung nach dem folgenden Prinzip statt:

1. Zunächst erfolgt eine Soft-Decision Decodierung des Codes $\mathcal{B}^{(1)}$. Der entsprechende Decoder gibt jedoch nur Wahrscheinlichkeitswerte $\tilde{\underline{z}}_\tau^{(1)}$ für die Codebits $\underline{z}_\tau^{(1)}$ des ersten äußeren Codes aus. Diese stellen die Numerierung des ersten inneren Untercodes dar und sind durch den Code $\mathcal{A}^{(1)}$ geschützt.
   Dieser Code wird nun decodiert, das heißt es werden, falls möglich, Fehler aus der ersten Numerierung entfernt. Der Decoder von $\mathcal{A}^{(1)}$ führt eine Hard-Decision für

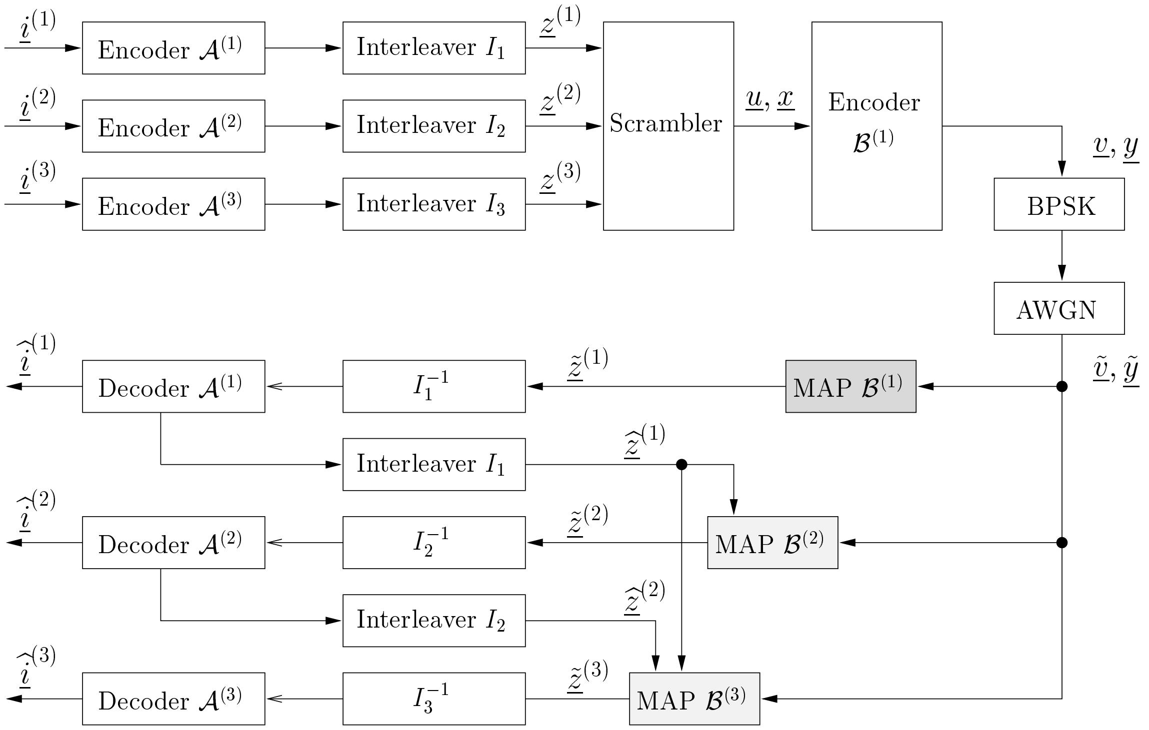


Abbildung 4.1: Gesamtsystem bei verallgemeinerter Codeverkettung mit Faltungscodes

Informations- und Codebits aus. Die Informationsfolge $\widehat{\underline{i}}^{(1)}$ wird ausgegeben und die Codefolge $\widehat{\underline{z}}^{(1)}$ wird an den inneren Decoder der nächsten Stufe weitergegeben.

2. In der zweiten Stufe wird nun die Empfangsfolge unter der Annahme decodiert, daß die Entscheidung der ersten Stufe korrekt war, daß also $\widehat{\underline{z}}^{(1)} = \underline{z}^{(1)}$ gilt. Die Decodierung erfolgt daher im Untercode $\mathcal{B}^{(2)}_{\widehat{\underline{z}}^{(1)}}$ statt, der eine höhere freie Distanz und damit bessere Korrektureigenschaften besitzt als $\mathcal{B}^{(1)}$.
Diesmal werden nur Wahrscheinlichkeitswerte $\tilde{\underline{z}}_\tau^{(2)}$ für die Codebits $\underline{z}_\tau^{(2)}$ des zweiten äußeren Codes ausgegeben. Anschließend wird dieser Code $\mathcal{A}^{(2)}$ decodiert. Der Decoder liefert wieder Hard-Decision Werte für die entsprechenden Informations- und Codebits. Die einen werden ausgegeben, die anderen wieder an die nächste Stufe weitergeleitet.

3. In der jeder weiteren Partitionierungsstufe erfolgt die Decodierung der Empfangsfolge immer in dem Untercode, der durch die vorhergehenden Stufen bestimmt ist, also unter der Annahme, daß die Entscheidung der vorigen Stufen korrekt ist.

Zur Soft Decision Decodierung der inneren Untercodes wurde, wie bereits in der Einleitung gesagt, ein veränderter MAP-Decoder eingesetzt. Dies ist in Bild 4.1 bereits dargestellt. Prinzipiell wäre auch die Anwendung eines SOVA (*Soft Output Viterbi Algorithm*) denkbar (siehe auch Kapitel 6). Eine Terminierung der inneren Codes wäre dann nicht erforderlich.

Zur Hard-Decision Decodierung der äußeren Codes wurden Viterbi-Decoder verwendet, die sowohl Informations- als auch Codebits ausgeben. Auch hier wäre der Einsatz eines MAP-Decoders denkbar.

Im folgenden wird näher auf die MAP-Decodierung eingegangen.

4.2 MAP-Decodierung der inneren Untercodes

Setzt man zur Partitionierung eines Faltungscodes einen Scrambler nach Abschnitt 3.5 ein, so können die resultierenden Untercodes mit Hilfe eines MAP-Decoders im konventionellen Trellis decodiert werden. Hierzu sind eine veränderte Terminierung und einige Modifikationen des MAP-Algorithmus erforderlich, auf die im folgenden eingegangen wird. Für weitergehende Betrachtungen zum MAP-Algorithmus nach Bahl u.a. sei beispielsweise auf [Fü96] und [Sch95] verwiesen.

4.2.1 Terminierung von Scrambler und innerem Code

Zur Decodierung mittels MAP-Decoder ist es nötig für den inneren Code eine Terminierung durchzuführen. Ist die Gedächtnislänge des äquivalenten Faltungscodes, den man durch die Verwendung eines Scramblers erhält, größer als die der ursprünglichen Codes, so sind dementsprechend mehr Terminierungsbits notwendig.
Die Anzahl der benötigten Terminierungsbits bei Verwendung eines Faltungsscramblers mit der Gedächtnislänge m_{Scr} beträgt:

$$\begin{aligned} l_{\mathrm{Term}} &= m_{\mathrm{Scr}} \cdot K + M \cdot K && (4.1)\\ &= m_{\mathrm{Scr}} \cdot K + K && (4.2) \end{aligned}$$

Prinzipiell kann man die Wirkung der Terminierung bezüglich Scrambler und innerem Faltungscode auch getrennt betrachten: Die ersten $m_{\mathrm{Scr}} \cdot K$ Nullen dienen der Terminierung des Scramblers (*Scramblerterminierung*). Durch die restlichen K Terminierungs-Nullen wird auch der innere Code terminiert (*Codeterminierung*).

Im Vergleich zur Partitionierung ohne Scrambler bewirkt die größere Terminierungslänge bei gleicher Blocklänge L einen gewissen Ratenverlust ΔR. Für große Blocklängen L kann man diesen jedoch vernachlässigen.

4.2.2 MAP nach Bahl, Cocke, Jelinek und Raviv

Zunächst soll der MAP-Algorithmus nach [BCJR74] in der bisher verwendeten Schreibweise angegeben werden, um darauf aufbauend die Änderungen beschreiben zu können, die für die Decodierung der Untercodes eines partitionierten Faltungscodes nötig sind. Dabei soll soweit möglich auch die Schreibweise von [BCJR74] beibehalten werden.

Hierzu benötigt man die UM-Darstellung der Informationsfolge

$$\underline{x} = (\underline{x}_0, \dots, \underline{x}_{L_{\mathrm{um}}-1}), \tag{4.3}$$

und die empfangene Folge in konventioneller Darstellung

$$\underline{\tilde{v}} = (\underline{\tilde{v}}_0, \dots, \underline{\tilde{v}}_{L-1}). \tag{4.4}$$

Dabei bezeichnet L nach Abschnitt 2.1.5 die Blocklänge im konventionellen Trellis und L_{um} die Blocklänge im UM-Trellis:

$$L = K \cdot L_{\mathrm{um}} \tag{4.5}$$

$\mathbf{V}$ bezeichnet im folgenden die Menge aller möglichen Übergänge $\underline{\gamma}_t$ und $\mathbf{V}'$ die Menge aller Zustände θ_t im konventionellen Trellis. Mit

$$(\underline{m}' \to \underline{m}) \tag{4.6}$$

soll in diesem Abschnitt der Übergang $\underline{\gamma}_t$ bezeichnet werden, der im Zustand $\underline{\theta}_{t-1} = \underline{m}'$ startet und im Zustand $\underline{\theta}_t = \underline{m}$ endet.

Außerdem werden mit P(A|B) bedingte Wahrscheinlichkeiten und mit P(A;B) Verbundwahrscheinlichkeiten beschrieben.

Die Wahrscheinlichkeitsfunktionen aus [BCJR74] kann man dann folgendermaßen darstellen:

- Wahrscheinlichkeiten der Vorwärtsrekursion:

$$\alpha_t(\underline{m}) = \mathrm{P}\left(\underline{\theta}_t = \underline{m}\ ;\ (\underline{\tilde{v}}_0, \ldots, \underline{\tilde{v}}_{t-1})\right) \tag{4.7}$$

- Wahrscheinlichkeiten der Rückwärtsrekursion:

$$\beta_t(\underline{m}') = \mathrm{P}\left((\underline{\tilde{v}}_t, \ldots, \underline{\tilde{v}}_{L-1})\ \middle|\ \underline{\theta}_t = \underline{m}'\right) \tag{4.8}$$

- Die bedingten Übergangswahrscheinlichkeiten sollen hier abweichend von [BCJR74] mit ξ_t statt mit γ_t bezeichnet werden, um eine Verwechslung mit den Übergängen $\underline{\gamma}_t$ zu vermeiden:

$$\xi_t(\underline{m}', \underline{m}) = \xi_t(\underline{\gamma}_t) = \mathrm{P}\left(\underline{\theta}_t = \underline{m}\ ;\ \underline{\tilde{v}}_{t-1}\ \middle|\ \underline{\theta}_{t-1} = \underline{m}'\right) \tag{4.9}$$

  Auf die Berechnung der Werte ξ_t soll nicht näher eingegangen werden, da ihre Berechnung auch für den veränderten MAP-Algorithmus im folgenden Abschnitt exakt gleich erfolgen kann wie in [BCJR74] beschrieben.

- Die Übergangswahrscheinlichkeiten:

$$\sigma_t(\underline{m}', \underline{m}) = \sigma_t(\underline{\gamma}_t) = \mathrm{P}\left(\underline{\theta}_{t-1} = \underline{m}'\ ;\ \underline{\theta}_t = \underline{m}\ ;\ \underline{\tilde{v}}\right) \tag{4.10}$$

Die Berechnung dieser Wahrscheinlichkeiten geschieht folgendermaßen:

Zunächst findet eine Initialisierung statt:

$$\alpha_0(\underline{0}) = 1, \qquad \text{und} \qquad \alpha_0(\underline{m}) = 0 \qquad \text{für} \quad \underline{m} \neq \underline{0} \tag{4.11}$$

$$\beta_L(\underline{0}) = 1, \qquad \text{und} \qquad \beta_L(\underline{m}') = 0 \quad \text{für} \quad \underline{m}' \neq \underline{0} \tag{4.12}$$

Dann werden in Abhängigkeit von der Empfangsfolge $\underline{\tilde{v}}$ die bedingten Übergangswahrscheinlichkeiten ξ_t berechnet (siehe [BCJR74], [And94]). Mit deren Hilfe können die Werte für α_t und β_t rekursiv berechnet werden:

$$\alpha_t(\underline{m}) = \sum_{\underline{m} \in \mathbf{V}'} \alpha_{t-1}(\underline{m}') \cdot \xi_t(\underline{m}', \underline{m}) \tag{4.13}$$

$$\beta_t(\underline{m}') = \sum_{\underline{m} \in \mathbf{V}'} \xi_{t+1}(\underline{m}', \underline{m}) \cdot \beta_{t+1}(\underline{m}) \tag{4.14}$$

Dies entspricht dem Durchlaufen des Trellis in Vorwärts- und Rückwärtsrichtung.

Die Übergangswahrscheinlichkeiten berechnen sich folgendermaßen:

$$\sigma_t(\underline{m}', \underline{m}) = \alpha_{t-1}(\underline{m}') \cdot \xi_t(\underline{m}', \underline{m}) \cdot \beta_t(\underline{m}) \tag{4.15}$$

Mit ihrer Hilfe werden schließlich die *a-posteriori* Wahrscheinlichkeiten $\underline{\tilde{x}}_\tau^{(i)}$ berechnet, mit denen die einzelnen Informationsbit den Wert 1 besitzen:

$$\underline{\tilde{x}}_\tau^{(i)} = \mathrm{P}\left(\underline{z}_\tau^{(i)} = 1 \;\middle|\; \underline{\tilde{v}}\right) = \frac{\sum\limits_{\underline{\gamma}_t \in \mathbf{B}} \sigma_t(\underline{\gamma}_t)}{\sum\limits_{\underline{\gamma}_t \in \mathbf{V}} \sigma_t(\underline{\gamma}_t)} \tag{4.16}$$

$$\text{mit} \qquad t = K\tau + i) \tag{4.17}$$

$$\text{und} \qquad \mathbf{B} = \left\{\underline{\gamma}_t \in \mathbf{V} \;\middle|\; \underline{\gamma}_t \cdot (0, \dots, 0, 1)^{\mathrm{T}} = 1\right\} \tag{4.18}$$

4.2.3 MAP zur Decodierung der inneren Untercodes

Zur Decodierung der inneren Untercodes bei verallgemeinerter Verkettung mit einem Scrambler sind einige Änderungen am oben beschriebenen MAP-Algorithmus vorzunehmen:

Das Mapping der Informationsbits bzw. dessen Umkehrung mit Hilfe der inversen Scramblermatrix $M^{-1}(D)$ nach Bild 3.12 und Gleichung 3.84 muß immer innerhalb des MAP-Decoders erfolgen. Ein Rückmapping nach dem MAP-Decoder ist nicht möglich, da dieser keine Bits sondern Soft-Informationen liefert, das Mapping jedoch auf binärer Algebra basiert.
Ab der zweiten Stufe müssen außerdem in Abhängigkeit von den bereits festgelegten Teilfolgen $\underline{\hat{z}}^{(1)}$ bestimmte Übergänge punktiert werden. (Der Sonderfall der Zustandspunktierung soll hier nicht getrennt betrachtet werden.)

Bei Verwendung eines Scramblers lautet die Informationsfolge in UM-Darstellung nach Gleichung 3.42:

$$\underline{z} = (\underline{z}_0, \dots, \underline{z}_{L-1}). \tag{4.19}$$

Das Mapping zwischen $\underline{z}$ und den Trellisübergängen $\underline{\gamma}_t$ kann man nach 3.84 beschreiben als

$$\underline{z}_\tau^{(i)} = \underline{\gamma}_t \cdot \underline{p}^{(i)} \qquad \text{für} \quad t = K(\tau+1) - \Delta^{(i)}. \tag{4.20}$$

Berücksichtigt man diesen Zusammenhang, so ergeben sich für den MAP-Decoder die folgenden Änderungen:

1. Stufe :

Die Menge **B** der Übergänge, für die ein bestimmtes Informationsbit den Wert 1 besitzt geht durch das durch den Scrambler veränderte Mapping über in $\mathbf{B}^{(1)}$.

Ansonsten erfolgt die Berechnung der Wahrscheinlichkeitswerte wie oben:

$$\underline{\tilde{z}}_\tau^{(i)} = \mathrm{P}\left(\underline{z}_\tau^{(i)} = 1 \;\middle|\; \underline{\tilde{v}}\right) = \frac{\sum\limits_{\underline{\gamma}_t \in \mathbf{B}^{(1)}} \sigma_t(\underline{\gamma}_t)}{\sum\limits_{\underline{\gamma}_t \in \mathbf{V}} \sigma_t(\underline{\gamma}_t)} \tag{4.21}$$

$$\text{mit} \qquad t = K(\tau+1) - \Delta^{(1)}$$

$$\text{und} \qquad \mathbf{B^{(1)}} = \left\{\underline{\gamma}_t \in \mathbf{V} \;\middle|\; \underline{\gamma}_t \cdot \underline{p}^{(1)} = 1\right\} \tag{4.22}$$

i. **Stufe** $(i > 1)$ **:**

Um in den folgenden Decodierstufen im richtigen Untercode zu decodieren, der durch $\underline{\hat{z}}^{(1)}, \ldots, \underline{\hat{z}}^{(i-1)}$ bestimmt ist, ist eine etwas veränderte Berechnung der Werte α_t und β_t nötig:

$$\alpha_t(\underline{m}) = \frac{\sum\limits_{(\underline{m}' \to \underline{m}) \in \mathbf{A}^{(i)}} \alpha_{t-1}(\underline{m}') \cdot \xi_t(m', m)}{\sum\limits_{\underline{m}'' \in \mathbf{V}} \sum\limits_{(\underline{m}' \to \underline{m}'') \in \mathbf{A}^{(i)}} \alpha_{t-1}(\underline{m}') \cdot \xi_t(m', m'')} \tag{4.23}$$

$$\beta_t(\underline{m}') = \frac{\sum\limits_{(\underline{m}', \underline{m}) \in \mathbf{A}^{(i)}} \xi_{t+1}(\underline{m}', \underline{m}) \cdot \beta_{t+1}(m)}{\sum\limits_{\underline{m}'' \in \mathbf{V}} \sum\limits_{(\underline{m}' \to \underline{m}'') \in \mathbf{A}^{(i)}} \xi_{t+1}(\underline{m}', \underline{m}'') \cdot \beta_{t+1}(m'')} \tag{4.24}$$

$$\text{mit} \qquad \mathbf{J}^{(i)} = \left\{j \;\middle|\; 0 \le j < i \quad \wedge \quad \Delta^{(j)} = \Delta^{(i)}\right\} \tag{4.25}$$

$$\text{und} \qquad \mathbf{A}^{(i)} = \left\{\underline{\gamma}_t \in \mathbf{V} \;\middle|\; \gamma_t \cdot \underline{p}^{(j)} = \underline{\hat{z}}_\tau^{(j)} \quad \forall \quad j \in \mathbf{J}^{(i)}\right\} \tag{4.26}$$

$A^{(i)}$ stellt dabei die Menge der im Untercode zulässigen Trellisübergänge dar. Da nicht mehr alle Übergänge in die Berechnung von $\underline{\alpha}_t$ bzw. $\underline{\beta}_t$ eingehen, muß durch den Nenner in Gleichung 4.23 und 4.24 dafür gesorgt werden, daß

$$\sum_{\underline{m} \in \mathbf{V}} \alpha_t(\underline{m}) = 1 \qquad \text{und} \qquad \sum_{\underline{m} \in \mathbf{V}} \beta_t(\underline{m}) = 1 \tag{4.27}$$

erfüllt ist. Beim ursprünglichen MAP-Algorithmus ist diese Bedingung automatisch erfüllt und daher keine Normierung erforderlich (siehe 4.13 und 4.14 bzw. [BCJR74]).

Berechnet man die Werte $\underline{\sigma}_t(\underline{\gamma}_t)$ unverändert nach Gleichung 4.15, so lauten die Wahrscheinlichkeiten für die einzelnen Informationsbits wie folgt:

$$\underline{\tilde{z}}_{\tau}^{(i)} = \mathrm{P}\left(\underline{z}_{\tau}^{(i)} = 1 \;\middle|\; \underline{\tilde{v}}\right) = \frac{\displaystyle\sum_{\underline{\gamma}_t \in \mathbf{B}^{(i)}} \sigma_t(\underline{\gamma}_t)}{\displaystyle\sum_{\underline{\gamma}_t \in \mathbf{A}^{(i)}} \sigma_t(\underline{\gamma}_t)} \tag{4.28}$$

mit $\quad t = K(\tau + 1) - \Delta^{(i)}$

und $$\mathbf{B}^{(i)} = \left\{\underline{\gamma}_t \in \mathbf{A}^{(i)} \;\middle|\; \underline{\gamma}_t \cdot \underline{p}^{(i)} = 1\right\} \tag{4.29}$$

Im Rahmen der vorliegenden Arbeit wurden beide beschriebenen MAP-Algorithmen als Module für das Simulationsprogramm `COSSAP` implementiert (siehe Anhang B.3). Alle in Kapitel 5 dargestellten Simulationen wurden mit Hilfe dieser Module durchgeführt.

KAPITEL

5

Verallgemeinert verkettete Faltungscodes

Mit den Faltungsscramblern nach Abschnitt 3.5.3 bzw. 3.5.2 wurde eine Möglichkeit zur optimalen Partitionierung von Faltungscodes bezüglich der freien Distanzen der Untercodes gefunden. In Kapitel 4 wurde dann die Grundlage zur Decodierung eines inneren Faltungscodes bei verallgemeinerter Verkettung in Form zweier modifizierter MAP-Algorithmen geschaffen.
In diesem Kapitel wird nun beschrieben, wie man darauf aufbauend einen inneren Faltungscode mit äußeren Faltungscodes verketten kann und welche Bitfehlerraten sich mit den neuen Konstruktionen ergeben. Dabei wird in erster Linie untersucht, welchen zusätzlichen Codierungsgewinn man durch die optimale Partitionierung mit einem Scrambler gegenüber GC-Codes mit zufällig partitionierten inneren Faltungscodes erzielt, die auch denkbar sind. Alle Überlegungen sollen anhand eines Beispiels erfolgen.

In Abschnitt 5.1 werden zunächst die Bitfehlerraten der Untercodes des inneren Faltungscodes betrachtet und mit den Bitfehlerraten bei zufälliger Partitionierung verglichen.

In Abschnitt 5.2 wird eine Methode zur geeigneten Wahl der äußeren Faltungscodes bei gegebenem inneren Code vorgestellt.

Ein erster Vergleich der erzielbaren Bitfehlerraten der GC-Codes mit und ohne Scrambler erfolgt dann in Abschnitt 5.3 anhand von Kurven, die sich aus der Konstruktion ergeben und für die keine weiteren Simulationen nötig sind.

Zur Bestätigung dieser ersten Abschätzungen wird in Abschnitt 5.4 ein Gesamtsystem mit geeigneten Blocklängen und Interleavern berechnet und simuliert.

5.1 Simulation der innereren Untercodes

Vergleicht man verschiedene Faltungscodes gleicher Rate im Bezug auf die erzielten Bitfehlerraten bei BPSK-Übertragung über einen AWGN-Kanal, so stellt man fest, daß man mit Codes mit höherer freier Distanz bessere Ergebnisse erzielt.

Für die Partitionierung von Faltungscodes bedeutet dies, daß die Untercodes, die man bei Partitionierung mit einem Scrambler erhält, im allgemeinen bei gleichem Signal-Rauschverhältnis geringere Bitfehlerraten nach der Decodierung erzielen als die Untercodes, die aus der zufälligen Partitionierung entstehen. Je größer die erzielte Distanzerhöhung ist, umso stärker ist dieser Effekt. Für eine Übertragung über „gute“ AWGN-Kanäle, also große Werte E_s/N_0, kann man die Bitfehlerrate eines Codes über seine freie Distanz d abschätzen und somit auch den zusätzlichen Codierungsgewinn $\Delta E_{s[\mathrm{dB}]}$, den man durch eine Distanzerhöhung von d_1 auf d_2 erzielt (vgl.[Pro89], S. 460 ff). Die Abschätzung für den zusätzlichen Codierungsgewinn lautet:

$$\Delta E_{s[\mathrm{dB}]} \approx 10 \log \left(\frac{d_2}{d_1} \right) \tag{5.1}$$

Die Bitfehlerrate eines Untercodes $\mathcal{B}^{(i)}_{\underline{z}^{(1)},\dots,\underline{z}^{(i-1)}}$ kann man mit Hilfe des Decoders aus Kapitel 4.2 simulieren, wenn man ihm statt der decodierten Folgen $\widehat{\underline{z}}^{(1)},\dots,\widehat{\underline{z}}^{(i-1)}$ der vorigen Stufe die wirklich gesendeten Informationsfolgen $\underline{z}^{(1)},\dots,\underline{z}^{(i-1)}$, also die korrekte Numerierung des Untercodes, zur Verfügung stellt.

Als Beispiel soll der auf Rate 4/5 punktierte Code $(23, 35)$ aus [YKH84] betrachtet werden (Punktierungsmatrix siehe Seite 84). Im Anhang sind in Tabelle A.2 die besten Block-, Diagonal- und UM-Scrambler aufgeführt, die durch systematische Suche nach Abschnitt 3.6.3 für diesen Code gefunden wurden. Für die Untercodes erzielt man bei Partitionierung mit diesen Scramblern die folgenden freien Distanzen:

Art des Scramblers	$d^{(1)}$	$d^{(2)}$	$d^{(3)}$	$d^{(4)}$
-	3	3	3	5
BSs	3	4	6	8
DSs	3	4	5	8
UMSs	3	4	6	12
DSt	3	4	7	14
UMSt	**3**	**5**	**7**	**15**

Tabelle 5.1: Freie Distanzen der Untercodes bei Partitionierung des Codes $(23, 35;\ 4/5)$ mit verschiedenen Scramblern

Wie man sieht wird mit einem UM-Scrambler zur Übergangspunktierung (UMSt) die beste Partitionierung erzielt. Die Partitionierung mit Diagonal- und Blockscramblern sowie die Partitionierung durch Zustandspunktierung (z.B. mit UMSs) ist in Übereinstimmung zu den Überlegungen in Abschnitt 3.5 schlechter im Bezug auf die freien Distanzen.
Für alle diese Scrambler (außer DSt) wurden die Bitfehlerraten der innerenen Codes simuliert. Die Ergebnisse sind in den Abbildungen 5.1 bis 5.5 über E_s/N_0 aufgetragen. In Übereinstimmung mit Gleichung 5.1 erhält man bei Partitionierung mit dem UM-Scrambler für Übergangspunktierung (UMSt) den größten Codierungsgewinn für die einzelnen Untercodes. Der zusätzliche Gewinn gegenüber zufälliger Partitionierung müßte nach Abschätzung mit Gleichung 5.1 für die vierte Partitionierungsstufe

$$\Delta E_{s[\mathrm{dB}]} \approx 10 \log \left(\frac{15}{5} \right) = 4.8 \text{ dB}$$

betragen. Aus der Simulation (Bild 5.1 und 5.5) ergibt sich bei einer Bitfehlerrate von 10^{-5} eine Differenz von $2.6 \text{ dB} - (-1.8) \text{ dB} = 4.4 \text{ dB}$. Abschätzung und Simulation stimmen also für den betrachteten Wertebereich von E_s/N_0 bereits recht gut überein.

Bitfehlerraten P_b der Untercodes bei Partitionierung des Codes (23,35; 4/5)

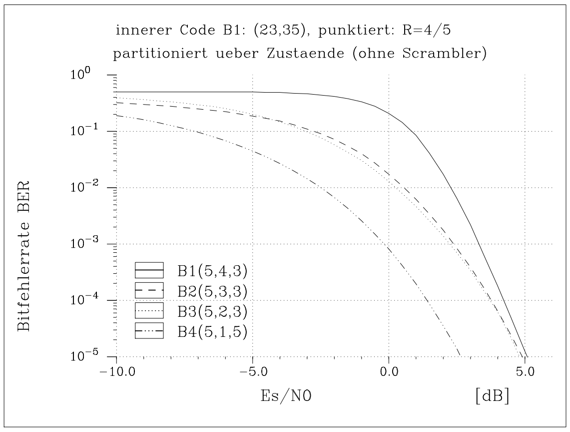


Abbildung 5.1: ohne Scrambler

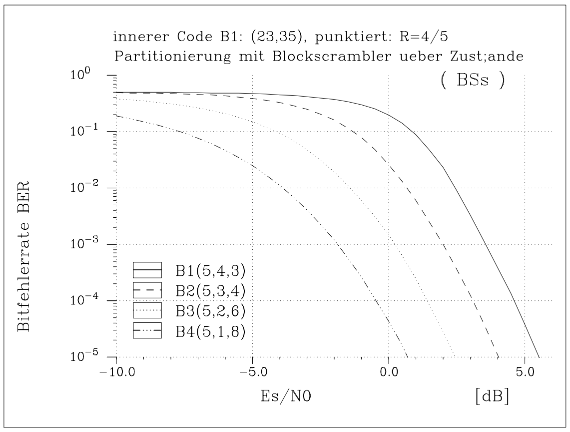


Abbildung 5.2: mit Blockscrambler für Zustandspunktierung (BSs)

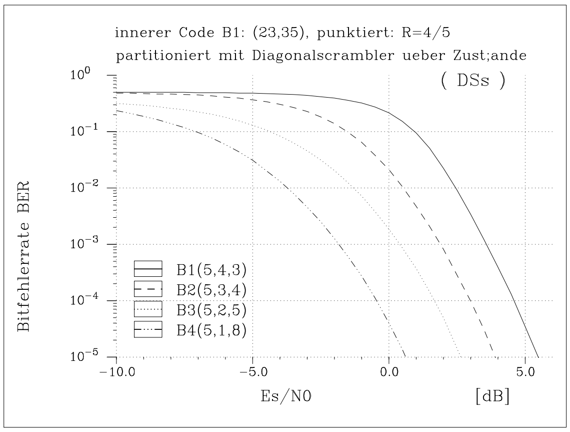


Abbildung 5.3: mit Diagonalscrambler für Zustandspunktierung (DSs)

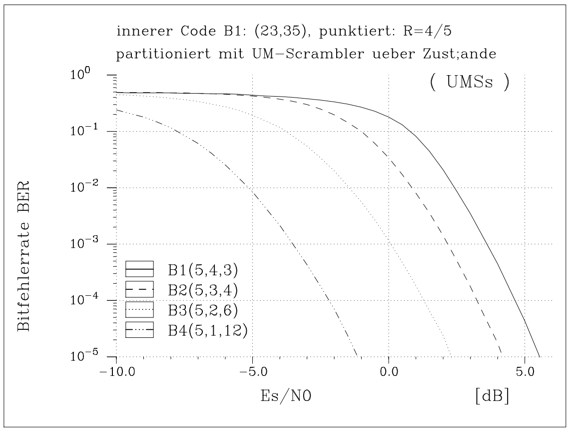


Abbildung 5.4: mit UM-Scrambler für Zustandspunktierung (UMSs)

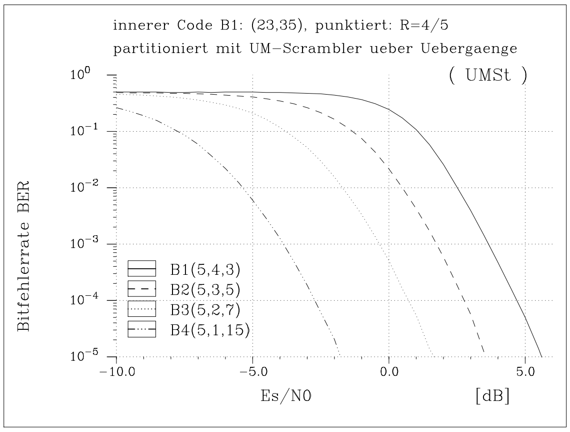


Abbildung 5.5: mit UM-Scrambler für Übergangspunktierung (UMSt)

Beim Vergleich der Bilder 5.1 bis 5.5 kann man außerdem folgenden Effekt beobachten: In der ersten Partitionierungsstufe tritt durch die Verwendung eines Scramblers ein geringer Verlust gegenüber dem ursprünglichen Code $\mathcal{B}^{(1)}$ auf. Dieser Effekt ist für alle Scrambler und Punktierungsarten ungefähr gleich groß und beträgt im Beispiel bei einer Bitfehlerrate von 10^{-5} etwa 0.5 dB.
Der Grund hierfür liegt wahrscheinlich im veränderten Mapping zwischen Code- und Informationsbits. Beim Rückmapping hängt jedes Informationsbit $\underline{z}_\tau^{(i)}$ von mehreren Bits $\underline{x}_\tau^{(i)}$ ab (vgl. Bild 3.12). Durch die Abbildung entstehen so zusätzliche Fehler.
Prinzipiell tritt dieser Effekt auch in allen anderen Stufen auf, doch überwiegt hier der zusätzliche Gewinn aufgrund der Distanzerhöhung. Da die Distanz der ersten Stufe nie erhöht werden kann wird der Effekt hier immer sichtbar.
Bei der Scramblersuche sollte man diesen Effekt soweit wie möglich einschränken, indem man bei der Konstruktion von $\underline{P}$ von mehreren gleichwertigen Spalten möglichst solche auswählt, die geringes Hamming-Gewicht besitzen (vgl Abschnitt 3.6.3). Der Effekt wird dann etwas minimiert.
Bei zufälliger Partitionierung kann man den Scrambler als Einheitsmatrix beschreiben. Der Effekt tritt hier nicht auf, da jedes Bit $\underline{z}_\tau^{(i)}$ nur von einem Bit $\underline{x}_\tau^{(i)}$ abhängt.

5.2 Auswahl der äußeren Codes

Die Auswahl der äußeren Codes bei verallgemeinerter Codeverkettung kann prinzipiell nach zwei verschiedenen Gesichtspunkten erfolgen:

1. **Optimierung der freien Distanz des Gesamtcodes:**

   Bei genügend großem Interleaving zwischen zwei verketteten Faltungscodes $\mathcal{A}^{(1)}$ und $\mathcal{B}^{(1)}$ kann man die freie Distanz d_{CC} des Gesamtcodes abschätzen über

   $$d_{\mathrm{CC}} \geq d_A^{(1)} \cdot d_B^{(1)}. \tag{5.2}$$

   Weitergehende Überlegungen zur Abschätzung der freien Distanz von verketteten Faltungscodes (insbesondere im Bezug auf die sogenannte *extended row distance* von Faltungscodes) findet man beispielsweise in [JZZ95],[JTZ88], [HJZ] und [DSV94].

   Die freie Distanz d_{GC} eines Codes, den man durch *verallgemeinerte* Verkettung erhält, kann man dementsprechend abschätzen über Gleichung 5.3 (siehe [BZ74]).

   $$d_{\mathrm{GC}} \geq \min\left\{d_A^{(1)} \cdot d_B^{(1)}, \ldots, d_A^{(K)} \cdot d_B^{(K)}\right\} \tag{5.3}$$

   Eine Möglichkeit der Codekonstruktion liegt nun darin, die äußeren Codes so zu wählen, daß die freien Distanzen aller Stufen identisch sind, daß also gilt:

   $$d_A^{(i)} \cdot d_B^{(i)} = d_A^{(j)} \cdot d_B^{(j)} \qquad \forall i \neq j \tag{5.4}$$

   Dieses Konstruktionsprinzip wird besonders häufig auf die Mindestdistanzen bei der Verkettung von Blockcodes angewandt (siehe z.B. [Bos92]) und führt im allgemeinen nicht zu einer Minimierung der Bitfehlerrate des Gesamtcodes $\mathcal{C}_{\mathrm{GC}}$ nach der Decodierung.

2. **Gleiche Bitfehlerrate in jeder Stufe:**
   Die zweite Möglichkeit besteht darin, die äußeren Codes so zu wählen, daß für einen bestimmten Arbeitspunkt, also für einen bestimmten Wert E_s/N_0, für jede Stufe die gleiche Bitfehlerrate $\mathrm{P_b}$ nach der Decodierung erreicht wird:

$$\mathrm{P}_{\mathrm{B,b}}^{(i)} = \mathrm{P_b} \qquad 1 \le i \le K \tag{5.5}$$

   Diese Bitfehlerrate wird im gegebenen Arbeitspunkt auch vom Gesamtcode $\mathcal{C}_{\mathrm{GC}}$ erzielt. Im allgemeinen Fall erfüllen die freien Distanzen der so bestimmten Codekombinationen nicht Gleichung 5.4.

In Rahmen der vorliegenden Arbeit wurde das zweite Konstruktionskriterium angewandt. Die Bestimmung der äußeren Codes nach diesem Kriterium soll daher näher erläutert werden. Man kann schrittweise wie folgt vorgehen:

- Man legt einen bestimmten Arbeitspunkt E_s/N_0 und die gewünschte Bitfehlerrate $\mathrm{P_b}$ für den Gesamtcode $\mathcal{C}_{\mathrm{GC}}$ fest.
- Anschließend bestimmt man die Bitfehlerraten $\mathrm{P}_{\mathrm{B,b}}^{(i)}, \ldots, \mathrm{P}_{\mathrm{B,b}}^{(i)}$ der inneren Codes $B^{(1)}, \ldots, B^{(K)}$ im Arbeitspunkt durch Simulation (oder Rechnung). Im Beispiel kann man sie aus den Abbildungen 5.1 bis 5.5 ablesen.
- Man wählt nun für die i-te Stufe den äußeren Code so, daß man bei einer Bitfehlerrate $\mathrm{P}_{\mathrm{B,b}}^{(i)}$ am Decodereingang am Ausgang die Bitfehlerrate $\mathrm{P_b}$ erhält.
  Hierbei geht man davon aus, daß die Eingangssymbole des äußeren Decoders wie bei BPSK-Übertragung über einen AWGN-Kanal statistisch unabhängig und gaußverteilt sind. Wählt man den Interleaver zwischen innerem und äußerem Code genügend groß und führt für den inneren Code eine Soft-Decision Decodierung durch, so ist diese Annahme sinnvoll.

Um die Auswahl des äußeren Codes wie beschrieben treffen zu können, ist es erforderlich für möglichst viel Codes mit Raten von 0 bis 1 die Eingangsbitfehlerrate zu kennen, die die gewünschte Ausgangsbitfehlerrate $\mathrm{P_b}$ zur Folge hat. Dazu muß man die Bitfehlerkurve all dieser Codes simulieren und für jeden die Eingangsfehlerrate $\mathrm{P_{B,b}}\,(\mathrm{P_b})$ nach Bild 5.6 bestimmen.

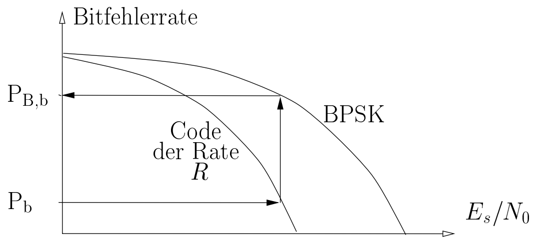


Abbildung 5.6: Bestimmung der Eingangsbitfehlerrate für eine gegebene Ausgangsbitfehlerrate

Trägt man diese „Eingangsbitfehlerraten“ $\mathrm{P_{B,b}}$ über der Rate der Codes auf, so erhält man für gegebenes $\mathrm{P_b}$ eine Kurve wie in Bild 5.7, aus der man direkt ermitteln kann, welche Rate der passende äußere Code für eine bestimmte Stufe besitzen muß.

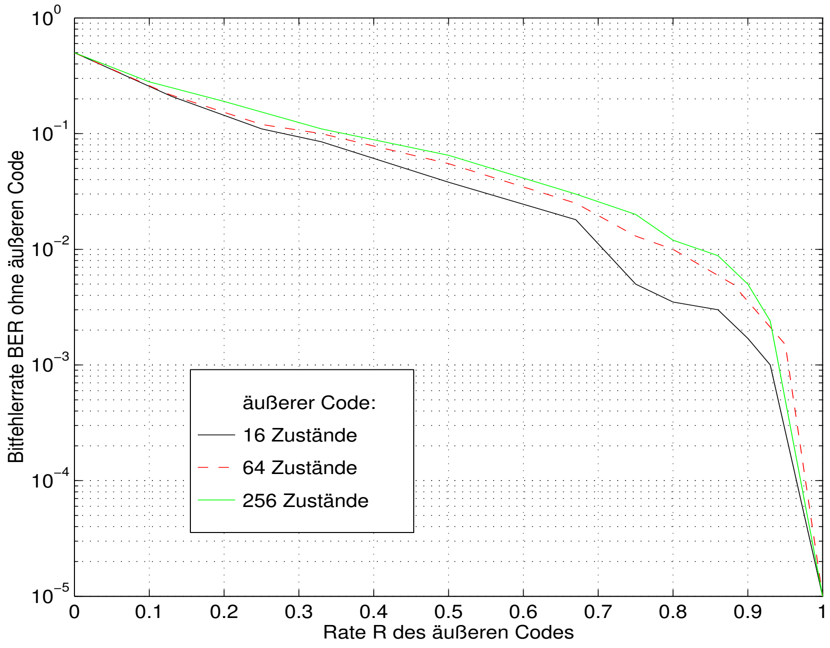


Abbildung 5.7: Äußere Codes für eine Gesamtbitfehlerrate von 10^{-5}

Man wählt hierzu die gegebene Bitfehlerrate $\mathrm{P}_{\mathrm{B,b}}^{(i)}$ auf der vertikalen Achse und liest die zugehörige Rate auf der horizontalen Achse ab. Aus der Menge der Faltungscodes, die zur Erstellung der Kurve verwendet wurden, wählt man denjenigen als äußeren Code $\mathcal{A}^{(i)}$, dessen Coderate am geringsten von der benötigten Rate abweicht.

Im vorliegenden Beispiel wurden Faltungscodes mit 16, 64 und 256 Zuständen aus [YKH84] und [Pal95] als äußere Codes verwendet. Davon sind die Codes mit 16 Zuständen in Anhang A.2 aufgelistet. Die Simulationsdaten zur Erstellung von Bild 5.7 wurden von H. Dieterich (AEG Mobile Communications, Ulm) zur Verfügung gestellt.
Darauf aufbauend wurde ein `Maple`-Programm zur automatischen Bestimmung geeigneter äußerer Codes erstellt (siehe Anhang B.2). Alle im folgenden verwendeten äußeren Codes wurden damit ermittelt.

5.3 Beurteilung des Gesamtcodes über Isoquanten

Ermittelt man die äußeren Codes nicht nur für einen Arbeitspunkt, sondern für eine ganze Reihe von Werten E_s/N_0, berechnet jeweils die Gesamtcoderate R des entstehenden GC-Codes und trägt diese über E_s/N_0 auf, so kann man eine erste Abschätzung der Performance des Gesamtcodes vornehmen. Auch wird ein Vergleich zwischen Codekonstruktionen mit und ohne Scrambler möglich, wenn man das Verfahren auf beide anwendet (siehe Bild 5.8 und 5.10).

Ein besserer Vergleich wird möglich, wenn man die ermittelten Coderaten statt über E_s/N_0 über dem normierten Signal-Rauschverhältnis E_b/N_0 aufträgt, welches man über den Zusammenhang

$$\frac{E_b}{N_0} = \frac{E_s}{N_0} \cdot \frac{1}{R} \tag{5.6}$$

für jeden der gewählten Arbeitspunkte des GC-Codes ermittelt. Auch dies ist mit dem oben erwähnten `Maple`-Programm möglich (siehe Anhang B.2).

Für den betrachteten Code (23, 35; 4/5) sind die resultierenden Kurven für Codekonstruktionen ohne Scrambler und mit UM-Scrambler (UMSt) in den Abbildungen 5.9 und 5.11 dargestellt. Die Bitfehlerrate $\mathrm{P_b}$ in den Arbeitspunkten wurde auf 10^{-5} festgelegt. Außerdem wurden immer äußere Codes mit gleicher Zustandszahl gewählt (16, 64 oder 64 Zustände).

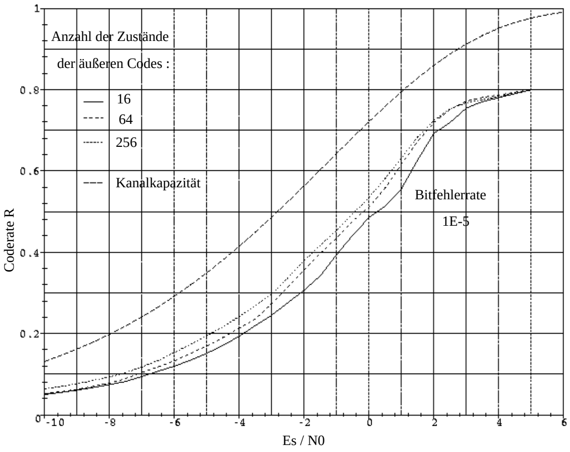


Abbildung 5.8: Coderate über E_s/N_0 für GC-Codes ohne Scrambler und $\mathcal{B}^{(1)} = (23, 35;\ 4/5)$

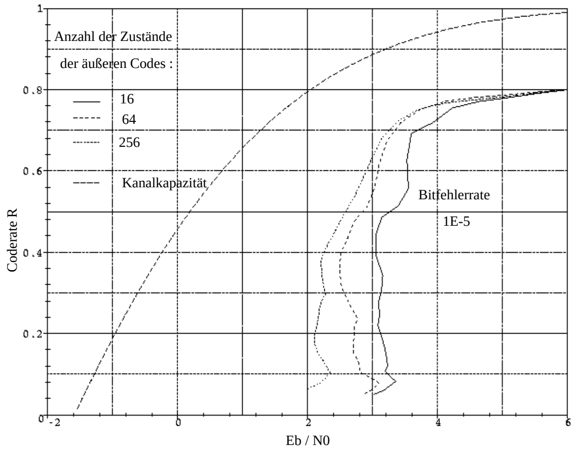


Abbildung 5.9: Isoquanten für GC-Codes ohne Scrambler und $\mathcal{B}^{(1)} = (23, 35;\ 4/5)$

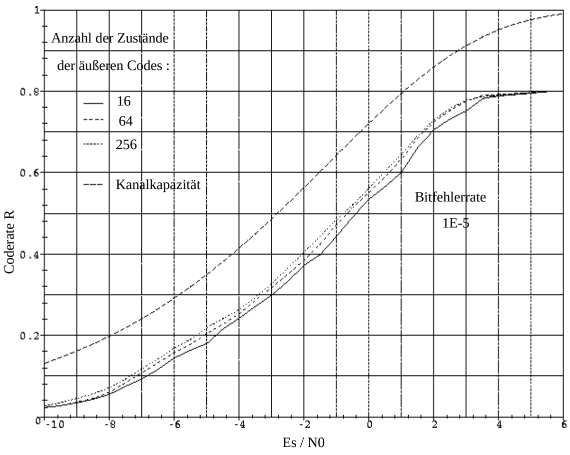


Abbildung 5.10: Coderate über E_s/N_0 für GC-Codes mit UMSt und $\mathcal{B}^{(1)} = (23, 35;\ 4/5)$

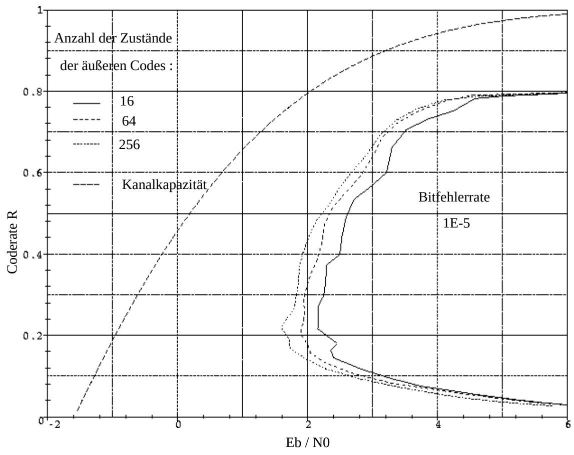


Abbildung 5.11: Isoquanten für GC-Codes mit UMSt und $\mathcal{B}^{(1)} = (23, 35;\ 4/5)$

In Analogie zu den *Isoquanten des Fehlerexponenten* (siehe [Kre89], S. 120 ff.) kann man auch diese Kurven als *Isoquanten* bezeichnen.
Zur Begründung: Denkt man sich die Bitfehlerrate $\mathrm{P_b}$, die sowohl von der Rate eines Codes als auch vom normierten Signal-Rauschverhältnis abhängt, in der dritten Dimension über R und E_b/N_0 aufgetragen, so stellen die Isoquanten eine Art Höhenlinien dar, da die Bitfehlerrate in allen Punkten der Linien konstant ist (im Beispiel $\mathrm{P_b} = 10^{-5}$).

Scrambler		Gesamtrate R	Raten der äußeren Codes $R_{\mathrm{A}}^{(1)}$	$R_{\mathrm{A}}^{(2)}$	$R_{\mathrm{A}}^{(3)}$	$R_{\mathrm{A}}^{(4)}$	$\mathrm{P_b} \leq 10^{-5}$ bei E_s/N_0 in dB	E_b/N_0 in dB
ohne	soll	0.394	0.061	0.488	0.551	0.870	-1.0	3.04
	ist	0.375	0	0.5	0.5	0.875		
mit UMSt	soll	0.406	0.029	0.239	0.764	1.0	-1.5	2.41
	ist	0.4	0	0.25	0.75	1.0		
ohne	soll	0.693	0.675	0.898	0.911	0.981	2.0	3.59
	ist	0.681	0.667	0.9	0.909	0.929		
mit UMSt	soll	0.707	0.588	0.938	1.0	1.0	2.0	3.52
	ist	0.686	0.5	0.929	1.0	1.0		

Tabelle 5.2: Raten der äußeren Codes $\mathcal{A}^{(i)}$, die sich bei der Gesamtcodekonstruktion mit dem inneren punktierten Code (23,35; 4/5) ergeben (mit und ohne Scrambler)

Die Isoquanten sind gut für einen Vergleich zwischen Codekonstruktionen mit und ohne Scrambler geeignet.

Wählt man eine bestimmte Coderate aus und vergleicht bei welchem Signal-Rauschverhältnis E_b/N_0 ein GC-Code mit dieser Rate die gewünschte Bitfehlerrate (im Beispiel 10^{-5}) erreicht, so erkennt man im Beispiel, daß dies bei der Verwendung des UM-Scramblers bei einem deutlich niedrigeren E_b/N_0 der Fall ist als ohne Scrambler.

In Tabelle 5.2 sind die Daten der konstruierten GC-Codes für die Raten 0.4 und 0.7 aufgeführt (äußere Codes mit 16 Zuständen). Bei einer Rate $R \approx 0.4$ erzielt man im Beispiel durch den Scrambler einen Gewinn von etwa 0.5 dB, (Wählt man äußere Codes mit größerer Gedächtnislänge, z.B. 64 oder 256 Zustände, so ist der durch den Scrambler erzielte Gewinn etwas geringer, siehe Bild 5.9 und 5.10.) Für $R \approx 0.7$ ist scheinbar kein Gewinn zu erwarten.

Eine andere Vergleichsmöglichkeit ist folgende: Man wählt ein bestimmtes Signal-Rauschverhältnis E_b/N_0 und liest ab, bei welcher Coderate man die gewünschte Bitfehlerrate erreicht.

Neben den Isoquanten ist in den Bildern 5.9 und 5.11 (und in 5.8 und 5.10) auch die Kanalkapazität für BPSK-Übertragung über den AWGN-Kanal eingezeichnet (zur Berechnung siehe [Pro89], [Kre89]).
Dadurch wird eine absolute Beurteilung der konstruierten GC-Codes ermöglicht. Im Beispiel ist man bei einer Rate von 0.4 unter Verwendung eines Scramblers und äußeren Codes mit 16 Zuständen etwa 2.7 dB von der Kanalkapazität entfernt (für $\mathrm{P_b} = 10^{-5}$).

5.4 Simulation mit äußeren Codes und Interleaver

Zur Überprüfung der Ergebnisse, die man aus den Isoquanten ablesen kann, die bei der Codekonstruktion entstehen, ist es im allgemeinen erforderlich Simulationen durchzuführen. In Rahmen der vorliegenden Arbeit wurde dies beispielsweise für die GC-Codes der Raten 0.4 und 0.7 aus Tabelle 5.2 durchgeführt (äußere Codes nach Tabelle A.5).

Prinzipiell ist es hierfür nötig die Blocklängen der verwendeten Faltungscodes festzulegen und geeignete Interleaver auszuwählen (vgl. Bild 4.1 auf Seite 58). Erst danach kann unter Berücksichtigung der Terminierungslängen die genaue Coderate des Gesamtsystems ermittelt werden.

Die Konstruktion eines verallgemeinert verketteten Codes geht also über die reine Bestimmung der äußeren Codes hinaus.

Generell sind verschiedene Gesamtkonstruktionen denkbar. Im folgenden wird eine Möglichkeit beschrieben, nach der man die einzelnen Komponenten eines GC-Codes aufeinander abstimmen kann. Zur Erklärung der dabei verwendeten Schreibweise sei auf den Abschnitt über terminierte Faltungscodes auf Seite 16 verwiesen.

Die Blocklängen aller äußeren Codes (in Codebits) sowie die Interleavinglängen aller K Blockinterleaver sollen gleich groß gewählt werden um statistische Unabhängigkeit zwischen den einzelnen Blöcken zu gewährleisten. Es soll also gelten:

$$l_{\mathrm{A,Code}}^{(i)} = l_{\mathrm{A,Code}}, \qquad i = 1, \ldots, K \tag{5.7}$$

$$b_{\mathrm{Z}}^{(i)} \cdot b_{\mathrm{S}}^{(i)} = l_{\mathrm{A,Code}} \tag{5.8}$$

Dabei wird mit $b_{\mathrm{Z}}^{(i)}$ die Zeilenzahl und mit $b_{\mathrm{S}}^{(i)}$ die Spaltenzahl eines Blockinterleavers bezeichnet. Bei der Wahl von $l_{\mathrm{A,Code}}$ muß man dann folgende Zusammenhänge beachten:

- Für die äußeren Codes $\mathcal{A}^{(i)}$ gilt:

$$l_{\mathrm{A,Info+}}^{(i)} = l_{\mathrm{A,Code}} \cdot 1/R_{\mathrm{A}}^{(i)} \tag{5.9}$$

- Für den inneren Code $\mathcal{B}$ gilt:

$$l_{\mathrm{B,Info}} = K \cdot l_{\mathrm{A,Code}} \tag{5.10}$$

$$l_{\mathrm{B,Term}} = m_{\mathrm{Scr}} \cdot K + K \tag{5.11}$$

$$l_{\mathrm{B,Info+}} = K \cdot l_{\mathrm{A,Code}} + m_{\mathrm{Scr}} \cdot K + K \tag{5.12}$$

$$l_{\mathrm{B,Code}} = l_{\mathrm{B,Info+}} \cdot 1/R_{\mathrm{B}} \tag{5.13}$$

Unter Berücksichtung dieser Zusammenhänge ist $l_{\mathrm{A,Code}}$ so zu wählen, daß sich aus Gleichung 5.9 und 5.13 ganzzahlige Werte ergeben.

Wurde ein geeignete Länge gefunden, dann kann man die Gesamtcoderate folgendermaßen berechnen:

$$R = \frac{l_{\mathrm{Info}}}{l_{\mathrm{B,Code}}} \tag{5.14}$$

Dabei gilt für die Anzahl der Informationsbits pro Block:

$$l_{\mathrm{Info}} = \sum_{i=1}^{K} l_{\mathrm{A,Info}}^{(i)} \tag{5.15}$$

$$l_{\mathrm{A,Info}}^{(i)} = l_{\mathrm{A,Info+}}^{(i)} - m_{\mathrm{A}} \tag{5.16}$$

Letztere Gleichung gilt nur, wenn alle äußeren Codes die gleiche Gedächtnislänge und damit auch die gleiche Terminierungslänge besitzen (und $k_{\mathrm{A}} = 1$ gilt; auch punktierte Codes):

$$l_{\mathrm{A,Term}}^{(i)} = m_{\mathrm{A}} \tag{5.17}$$

Wird ein innerer Untercode durch keinen äußeren Code geschützt ($R_{\mathrm{A}}^{(i)} = 1$), oder wird eine Stufe nicht zur Informationsübertragung genutzt ($R_{\mathrm{A}}^{(i)} = 0$), so müssen obige Gleichungen teilweise etwas abgeändert werden (da dann z.B. keine Terminierung nötig ist).

	mit UMSt				ohne Scrambler			
i	1	2	3	4	1	2	3	4
$b_{\text{Z}}^{(i)}$	–	125	100	25	–	125	100	25
$b_{\text{S}}^{(i)}$	–	40	50	200	–	40	50	200
$l_{\text{A,Code}}$	5000				5000			
$R_{\text{A}}^{(i)}$	0	1/4	3/4	1	0	1/2	1/2	7/8
$l_{\text{A,Info+}}^{(i)}$	0	1250	3750	5000	0	2500	2500	4375
$l_{\text{A,Term}} = m_{\text{A}}$	4				4			
${}^{(i)}_{\text{A,Info}}$	0	1246	3746	5000	0	2496	2496	4371
l_{Info}	9992				9363			
$l_{\text{B,Info}}$	20000				20000			
m_{Scr}	3				0			
$l_{\text{B,Term}}$	16				4			
$l_{\text{B,Info+}}$	20016				20004			
R_{B}	4/5				4/5			
$l_{\text{Code}} = l_{\text{B,Code}}$	25020				25005			
R	0.399				0.374			

Tabelle 5.3: Parameter eines GC-Codes mit $\mathcal{B}^{(1)} = (23, 35;\ 4/5)$ und $R \approx 0.4$

	mit UMSt				ohne Scrambler			
i	1	2	3	4	1	2	3	4
$b_{\text{Z}}^{(i)}$	132	165	110	70	132	165	110	70
$b_{\text{S}}^{(i)}$	35	28	42	66	35	28	42	66
$l_{\text{A,Code}}$	4620				4620			
$R_{\text{A}}^{(i)}$	1/2	13/14	1	1	2/3	9/10	10/11	13/14
$l_{\text{A,Info+}}^{(i)}$	2310	4290	4620	4620	3080	4158	4200	4290
$l_{\text{A,Term}} = m_{\text{A}}$	4				4			
${}^{(i)}_{\text{A,Info}}$	2306	4286	4616	4616	3076	4154	4196	4286
l_{Info}	15824				15712			
$l_{\text{B,Info}}$	18480				18480			
m_{Scr}	3				0			
$l_{\text{B,Term}}$	16				4			
$l_{\text{B,Info+}}$	18496				18484			
R_{B}	4/5				4/5			
$l_{\text{Code}} = l_{\text{B,Code}}$	23120				23105			
R	0.684				0.680			

Tabelle 5.4: Parameter eines GC-Codes mit $\mathcal{B}^{(1)} = (23, 35;\ 4/5)$ und $R \approx 0.7$

Für die Codes der Rate 0.4 und 0.7 nach Tabelle 5.2 wurden Simulationen mit den Parametern aus Tabelle 5.3 und 5.4 durchgeführt. Der prinzipielle Simulationsaufbau entsprach dabei Bild 4.1. Die Ergebnisse sind in den Abbildungen 5.12 bis 5.14 dargestellt.

Außerdem wurde in den Bildern 5.12 und 5.13 jeweils eine weitere Bitfehlerkurve c) aufgetragen. Diese wurde durch Simulation eines verketteten Codes ohne Scrambler erzielt. Dabei wurden die äußeren Codes des GC-Codes mit Scrambler verwendet (Kurve a). Betrachtet man Tabelle 5.2 so wird klar weshalb diese Kombination so schlechte Ergebnisse liefert. Die inneren Codebits der dritten bzw. vierten Stufe werden durch keine äußeren Codes geschützt, die Bitfehlerkurve c) verläuft daher für den Code der Rate 0.4 parallel zu der des inneren Codes $\mathcal{B}^{(4)}$ (ohne Scrambler) und für den Code der Rate 0.7 parallel zu der des inneren Codes $\mathcal{B}^{(3)}$. Anhand dieser Codes wird deutlich, was eine falsche Wahl der äußeren Codes bewirkt, sie sollen im weiteren nicht weiter betrachtet werden.

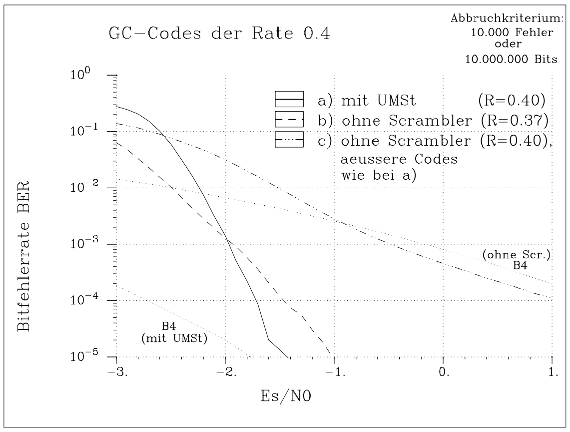


Abbildung 5.12: Bitfehlerraten von GC-Codes der Rate $R \approx 0.4$ mit und ohne Scrambler

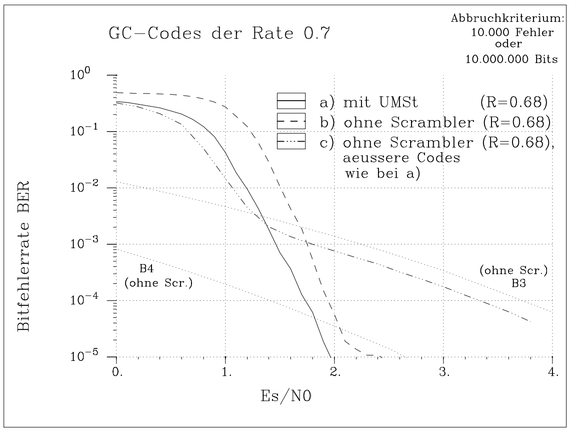


Abbildung 5.13: Bitfehlerraten von GC-Codes der Rate $R \approx 0.7$ mit und ohne Scrambler

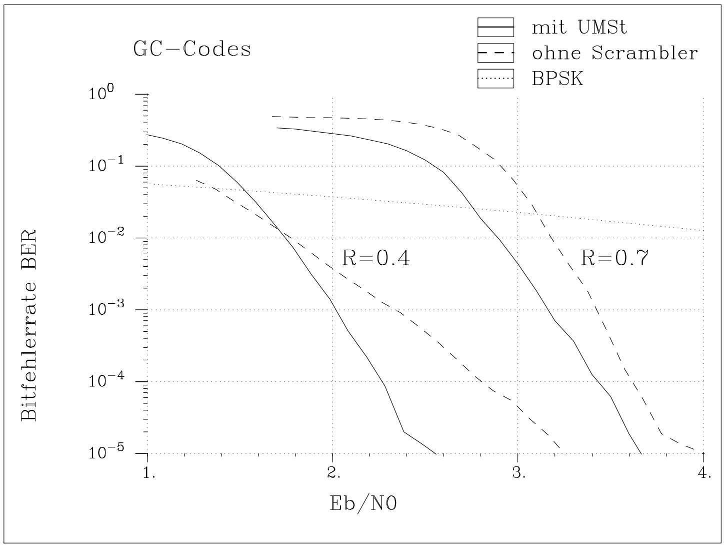


Abbildung 5.14: Bitfehlerraten von GC-Codes mit $\mathcal{B}^{(1)} = (23, 35;\ 4/5)$ über E_b/N_0

Betrachtet man die Bitfehlerraten der korrekt konstruierten Codes (Kurven a und b), so zeigt sich, daß sie tatsächlich weitgehend die Abschätzungen des vorigen Abschnitts 5.3 bestätigen:
In der Simulation erzielt man mit dem GC-Code der Rate 0.4 mit Scrambler (Kurve a in Bild 5.12) für $\mathrm{P_b} = 10^{-5}$ einen zusätzlichen Codierungsgewinn $\Delta E_{s,[\mathrm{dB}]}$ von ca. 0.4 dB gegenüber dem GC-Code mit zufälliger Partitionierung des inneren Codes (Kurve b). Die Abschätzung ergab 0.5 dB. Auch die Absolutwerte stimmen mit denen in Tabelle 5.2 gut überein. Der Gewinn bezüglich des normierten Signal-Rauschverhältnisses E_b/N_0 ist etwas größer (etwa 0.6 dB, siehe Bild5.14), da der GC-Code ohne Scrambler eine minimal geringere Rate besitzt.

Für die GC-Codes der Rate 0.7 in Bild 5.13 gilt folgendes: Der Code mit Scrambler bestätigt genau die Abschätzung nach Tabelle 5.2. Dagegen weicht der GC-Code ohne Scrambler von der Werten, die sich aus der Konstruktion ergeben ab und ist daher etwas schlechter als der GC-Code mit Scrambler.

Für das betrachtete Beispiel stimmen die Abschätzungen anhand der Isoquanten (bzw. die Werte aus der Konstruktion) insgesamt jedoch gut mit der Simulation überein.

Sowohl die Werte der Konstruktion (die Isoquanten) als auch die Simulation bestätigen, daß durch den Einsatz eines Scramblers ein zusätzlicher Gewinn ohne Erhöhung der Decodierkomplexität erzielt werden kann. Dessen Größe ist offenbar abhängig von der Wahl der Coderate des GC-Codes. Je näher sie an die Rate des inneren Codes heranrückt, umso geringer fällt der zusätzliche Gewinn durch den Scrambler aus. Dies wird auch durch die Ergebnisse der Simulationen und Abschätzungen in Anhang A.3 bestätigt, die für den *ESA-NASA Satellite Standard Code* (133,171) mit und ohne Scrambler durchgeführt wurden.
Es ist daher offensichtlich wichtig, möglichst hochratige Codes als innere Codes zu wählen.

Auch die absolute Größe des zusätzlichen Codierungsgewinns ist in jedem Fall vom inneren Code und der Anzahl der Partitionierungsstufen abhängig. Man vergleiche hierzu beispielsweise die Bilder 5.11 und A.6.

Abschließend sei nun noch auf einen Vergleich hingewiesen, der sehr eindrucksvoll den Unterschied zwischen der Wirkung eines Blockscramblers für Zustandspunktierung (BSs) und einem Unit-Memory Scrambler für Übergangspunktierung (UMSt) verdeutlicht. Man betrachte hierzu die für den Code $(133, 171)$ erzielten freien Distanzen in Tabelle A.1 im Anhang oder vergleiche die Bitfehlerraten der inneren Untercodes aus [BDS96a] mit denen in Bild A.2.
Der sehr große Unterschied bezüglich der Bitfehlerraten des inneren Codes bewirkt jedoch offenbar für den GC-Code (der Rate 0.4, vgl [BDS96a]) nur einen Gesamtgewinn in der Größenordnung des obigen Beispiels . Dies kann man aus den Isoquanten in Bild A.6 ablesen. Eine Simulation des GC-Codes mit innerem Code $(133, 117)$ und UM-Scrambler (UMSt) wurde im Rahmen dieser Arbeit nicht durchgeführt, würde dieses Ergebnis aber wahrscheinlich bestätigen.

KAPITEL

6

Zusammenfassung und Ausblick

Im Rahmen der vorliegenden Diplomarbeit wurden verschiedene Methoden zur Partitionierung von Faltungscodes untersucht und weiterentwickelt. Als Ausgangspunkt hierfür wurde eine zufällige Partitionierung mit Hilfe der Informationssequenz gewählt (Abschnitt 3.2). Die Partitionierung über die Informationssequenz wurde im folgenden beibehalten, aber durch die Verwendung sogenannter Block- und Diagonalscrambler nach [BDS96a] und [BDS96b] wurden Untercodes mit höheren freien Distanzen erzielt (Abschnitt 3.5.1 und 3.5.2). Die Wirkung dieser Scrambler wurde ausführlich untersucht. Dabei ergaben sich einige Möglichkeiten zur weiteren Verbesserung der verwendeten Scrambler. Zum einen konnten durch die Punktierung von Übergängen im Trellisdiagramm anstelle der zunächst verwendeten Zustandspunktierung größere Distanzerhöhungen für die Untercodes erzielt werden. Zum anderen ergab sich ein weiterer Gewinn durch die Verallgemeinerung der Scramblerstruktur (UM-Scrambler, Faltungsscrambler). Für diese neuen Scrambler wurde eine mathematische Beschreibung gefunden, welche die bisher bekannten Scrambler sowie die zufällige Partitionierung als Sonderfälle beinhaltet (Kapitel 3.5). Außerdem wurde ein Suchalgorithmus für die neuen Faltungsscrambler entwickelt (siehe Abschnitt 3.6.3) und implementiert. Die Implementierung (Anhang B.1) erlaubt auch die Einschränkung der Suche auf Block- und Diagonalscrambler sowie die Angabe der Punktierungsart, so daß ein Vergleich zwischen den einzelnen Scramblerarten möglich wurde.

Der zweite Hauptteil der Arbeit bestand in der Suche nach einem geeigneten Soft-Decision Decodieralgorithmus, der für die Decodierung der inneren Untercodes bei verallgemeinerter Codeverkettung von Faltungscodes eingesetzt werden kann, wenn man einen Scrambler zur Partitionierung verwendet. Prinzipiell wäre es möglich die inneren Untercodes als Unit Memory Codes aufzufassen und mit einem MAP-Decoder nach Bahl im UM-Trellis zu decodieren. Dies wäre jedoch mit einer wesentlich höheren Decodierkomplexität verbunden als die Decodierung im konventionellen Trellis des inneren Codes. Der Gewinn, den man durch einen Scrambler erzielt, würde diesen höheren Aufwand nicht unbedingt rechtfertigen (vgl. Abschnitt 5.4). Außerdem müßte der Algorithmus auch dann ab der zweiten Decodierstufe abgeändert werden. Aus diesem Grund wurde ein ein modifizierter MAP-Algorithmus formuliert und für das Simulationsprogramm `COSSAP` implementiert, der den Einsatz eines Scramblers zur Partitionierung erlaubt und die Decodierung aller Untercodes im konventionellen Trellis ohne Erhöhung der Decodierkomplexität realisiert (Abschnitt 4.2).

Im letzten Teil der Arbeit wurde unter Verwendung der neuen Komponenten (Scrambler und MAP-Decoder) ein Gesamtsystem zur Untersuchung der Leistungsfähigkeit von verall-

gemeinert verketteten Codes (GC-Codes) mit inneren und äußeren Faltungscodes realisiert. Zur Bestimmung geeigneter äußerer Codes wurde hierbei ein Verfahren verwendet, welches die Gesamtbitfehlerrate eines GC-Codes in einem bestimmten Arbeitspunkt optimiert (Abschnitt 5.2). In den meisten Veröffentlichungen zur verallgemeinerten Codeverkettung wurde bisher von einer Optimierung der freien Distanz des Gesamtcodes ausgegangen. Im allgemeinen erhält man dann jedoch keine Optimierung der Bitfehlerrate.
In den Abschnitten 5.3 und 5.4 wurde schließlich für ein Beispiel die Leistungsfähigkeit der verallgemeinerten Codeverkettung bei der Verwendung von Faltungscodes und Partitionierung mit einem Scrambler untersucht. Im Beispiel wurde durch den Scrambler ein zusätzlicher Gewinn von bis zu 0.6 dB gegenüber zufälliger Partitionierung erzielt. Dieser zunächst gering erscheinende Gewinn muß unter zwei Gesichtspunkten betrachtet werden. Zum einen kann er ohne wesentliche Erhöhung der Decodierkomplexität erreicht werden, zum anderen muß man die relativ geringe Entfernung von ca. 3 dB zur Kanalkapazität berücksichtigen. Allgemeinere Aussagen können erst getroffen werden wenn man mehrere GC-Codes auf ihre Leistungsfähigkeit hin untersucht hat (siehe auch Anhang A.1). Diese müssen dann auch mit verallgemeinert verketteten Blockcodes verglichen werden, was im Rahmen der vorliegenden Arbeit nicht mehr möglich war.

Zusammenfassend kann man sagen, daß die erzielten Ergebnisse in jedem Fall den anfänglichen Erwartungen entsprechen, insbesondere was die bei der Partitionierung erzielten freien Distanzen betrifft. Bezüglich der Gesamtkonstruktion der GC-Codes mit inneren und äußeren Faltungscodes sind sicher noch weitere Gewinne erzielbar, beispielsweise wenn man die Wahl der äußeren Codes oder das Interleaving näher untersucht.

Zum Abschluß soll noch ein kurzer Überblick über die relativ große Menge von denkbaren Verbesserungen, Weiterentwicklungen und Anwendungen gegeben werden, die sich aus der Möglichkeit der Partitionierung eines Faltungscodes ergeben. Denkbar sind folgende Themen und Fragen:

- Anwendung der Partitionierung auf Modulationsverfahren mit Gedächtnis zur Codierten Modulation.
- Partitionierung nichtlinearer Faltungscodes oder Modulationsverfahren.
- Partitionierung differentieller Modulationsverfahren mit Hilfe rekursiver Scrambler.
- Untersuchung von rekursiven Scramblern.
- Partitionierung von nichtterminierten Faltungscodes (Decodierung z.B. mit SOVA).
- Untersuchung des Konzepts der „longer branches“ (siehe [Die96]).
- Scrambler, die eine geringe Erhöhung der Decodierkomplexität erfordern, aber einen größeren Gewinn erzielen.
- Untersuchung der Frage, welche Faltungscodes sich gut zur Partitionierung eignen.
- Konstruktion von Scramblern im Hinblick auf andere Kriterien (z.B. *extended row distance*, Fehlergewicht, …).
- Wahl der äußeren Codes nach anderen Kriterien.
- Bessere Wahl der Interleaver zwischen inneren und äußeren Codes sowie der Blocklängen.

ANHANG

A

Ergebnisse

A.1 Scrambler zur Partitionierung bestimmter Faltungscodes

Art	inverser Scrambler $\underline{M}(D)^{-1}$	$d^{(1)},\dots,d^{(6)}$
UMSt	$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & D+1 & 0 \\ 0 & D+1 & 0 & 0 & D & 0 \\ 0 & D & 1 & D+1 & D & 0 \\ 0 & 0 & D+1 & 0 & 0 & 1 \\ D+1 & D & 0 & 0 & 0 & 1 \end{bmatrix}$	**10,10,10,16,24,50**
DSt	$\begin{bmatrix} D+1 & 1 & 1 & 1 & 1 & 0 \\ D & D+1 & 0 & 1 & 1 & 0 \\ D & 0 & D+1 & 0 & 0 & 1 \\ 0 & D & 0 & D+1 & 1 & 1 \\ 0 & D & D & D & D+1 & 1 \\ 0 & d & 0 & D & 0 & 1 \end{bmatrix}$	10,10,10,12,20,44
CSs	$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & D & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & D & 0 & D & 1 & 0 \\ 1 & 0 & 1 & 0 & D & 0 \\ 1 & D & D & D & 0 & 1 \end{bmatrix}$	10,10,10,14,20,40
DSs	$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ D & 1 & 1 & 0 & 0 & 1 \\ D & D & 1 & 0 & 1 & 1 \\ D & 0 & D & 1 & 0 & 1 \\ 0 & D & D & D & 1 & 1 \\ D & 0 & 0 & D & D & 1 \end{bmatrix}$	10,10,10,14,20,38
BSs	siehe [BDS96a]	10,10,10,12,12,16

Tabelle A.1: Beste inverse Scrambler für 6-fache Partitionierung des Code $(133, 171)$

Art	inverser Scrambler $\underline{M}(D)^{-1}$	Scrambler $\underline{M}(D)$	$d^{(1)},\dots,d^{(4)}$
UMSt	$\begin{bmatrix} 0 & 1 & D+1 & 0 \\ 1 & 1 & D & 0 \\ D+1 & 0 & D & 0 \\ 0 & D+1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} D & D & 1 & 0 \\ D^2 & (D+1)^2 & D+1 & 0 \\ D+1 & D+1 & 1 & 0 \\ D^2\,(D+1) & (D+1)^3 & (D+1)^2 & 1 \end{bmatrix}$	**3,5,7,15**
DSt	$\begin{bmatrix} D+1 & 1 & 1 & 1 \\ D & D+1 & 1 & 1 \\ 0 & D & D+1 & 0 \\ D & D & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & D^2+D+1 & D & D^2+D \\ 0 & D+1 & 1 & D+1 \\ 0 & D & 1 & D \\ D & D^3 & (D+1)\,D & D^3+D+1 \end{bmatrix}$	3,4,7,14
UMSs	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & D & 1 & 1 \\ 1 & D & D & 0 \\ D & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} D\,(D+1) & D & 1 & D \\ D^2+D+1 & D & 1 & D \\ D^2 & D+1 & 1 & D+1 \\ D^2\,(D+1) & D^2 & D & D^2+1 \end{bmatrix}$	3,4,6,12
DSs	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ D & 0 & 1 & 0 \\ 0 & D & D & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ D & D & 1 & 0 \\ D^2 & D^2+D & D & 1 \end{bmatrix}$	3,4,5,8
BSs	$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$	3,4,6,8

Tabelle A.2: Scrambler für 4-fache Partitionierung des punktierten Codes (23,35; 4/5)

Art	inverser Scrambler $\underline{M}(D)^{-1}$	Scrambler $\underline{M}(D)$	$d^{(1)},d^{(2)},d^{(3)}$
UMSt	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & D & 1 \end{bmatrix},\dots$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ D & D+1 & 1 \end{bmatrix},\dots$	$\mathbf{5,6,8}$
DSt	$\begin{bmatrix} 1 & 1 & 0 \\ D & 1 & 1 \\ D & 0 & 1 \end{bmatrix},\begin{bmatrix} 1 & 1 & 1 \\ D & 1 & 0 \\ D & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ D & D & 1+D \end{bmatrix},\begin{bmatrix} 1 & 1 & 1 \\ D & 1+D & D \\ D & D & 1+D \end{bmatrix}$	$5,6,8$
BSt	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix},\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},\dots$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix},\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},\dots$	$5,6,6$
UMSs	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & D & 0 \end{bmatrix},\dots$	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},\begin{bmatrix} D & 0 & 1 \\ 1 & 0 & 0 \\ D & 1 & 1 \end{bmatrix},\dots$	$5,5,7$
DSs	$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ D & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ D & D & 1 \end{bmatrix}$	$5,5,7$

Tabelle A.3: Beste Scrambler für 3-fache Partitionierung des Code $(5,7)$

Art	inverser Scrambler $\underline{M}(D)^{-1}$	Scrambler $\underline{M}(D)$	$d^{(1)}, d^{(2)}$
UMSt	$\begin{bmatrix} 1 & 0 \\ D+1 & 1 \end{bmatrix}, \ldots$	$\begin{bmatrix} 1 & 0 \\ D+1 & 1 \end{bmatrix}, \ldots$	$\mathbf{5, 7}$
$UMSs$	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \ldots$	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \ldots$	$5, 6$
DSs	$\begin{bmatrix} 1 & 0 \\ D & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ D & 1 \end{bmatrix}$	$5, 6$
BSs	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$	$5, 6$

Tabelle A.4: Beste Scrambler für 2-fache Partitionierung des Code $(5, 7)$

A.2 Verwendete äußere Codes (16 Zustände)

Coderate R		d_{free}	Generatorpolynome (oktal)	punktiert	Quelle
0.0625	1/16	64	$25^4, 27, 33^4, 35^3, 37^4$		[Pal95]
0.0667	1/15	60	$25^4, 27, 33^4, 35^2, 37^4$		[Pal95]
0.0714	1/14	56	$25^3, 27^2, 33^3, 35^3, 37^3$		[Pal95]
0.0769	1/13	52	$25^3, 27^2, 33^3, 35^2, 37^3$		[Pal95]
0.0833	1/12	48	$25^3, 27, 33^3, 35^2, 37^3$		[Pal95]
0.0909	1/11	44	$25^3, 27, 33^3, 35, 37^3$		[Pal95]
0.1	1/10	40	$25^3, 33^3, 35, 37^3$		[Pal95]
0.1111	1/9	36	$25^2, 27, 33^2, 35^2, 37^2$		[Pal95]
0.125	1/8	32	$25^2, 27, 33^2, 35, 37^2$		[Pal95]
0.1429	1/7	28	$25, 27^2, 33, 35^2, 37$		[Pal95]
0.1667	1/6	24	$25, 27, 33, 35^2, 37$		[Pal95]
0.2	1/5	20	$25, 27, 33, 35, 37$		[Pal95]
0.25	1/4	16	$25, 27, 33, 37$		[Pal95]
0.3333	1/3	12	$25, 33, 37$		[Pal95]
0.5	1/2	7	$23, 35$		[Pal95],[YKH84]
0.6667	2/3	4	$23, 25$	x	[YKH84]
0.75	3/4	3	$23, 35$	x	[YKH84]
0.8	4/5	3	$23, 35$	x	[YKH84]
0.833	5/6	3	$23, 35$	x	[YKH84]
0.8571	6/7	3	$23, 35$	x	[YKH84]
0.875	7/8	3	$23, 35$	x	[YKH84]
0.8889	8/9	3	$23, 35$	x	[YKH84]
0.9	9/10	2	$23, 35$	x	[YKH84]
0.9091	10/11	2	$23, 35$	x	[YKH84]
0.9167	11/12	2	$23, 35$	x	[YKH84]
0.9231	12/13	2	$23, 35$	x	[YKH84]
0.9286	13/14	2	$23, 35$	x	[YKH84]

Tabelle A.5: Faltungscodes mit 16 Zuständen, die als äußere Codes verwendet wurden

A.3 Partitionierung des ESA-NASA-Codes (133,171)

(Siehe folgende Seite.)

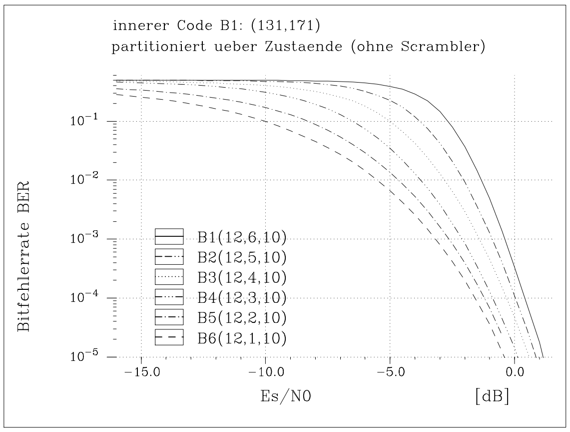


Abbildung A.1: Bitfehlerraten der Untercodes bei zufälliger Partitionierung des Codes $(133, 171)$

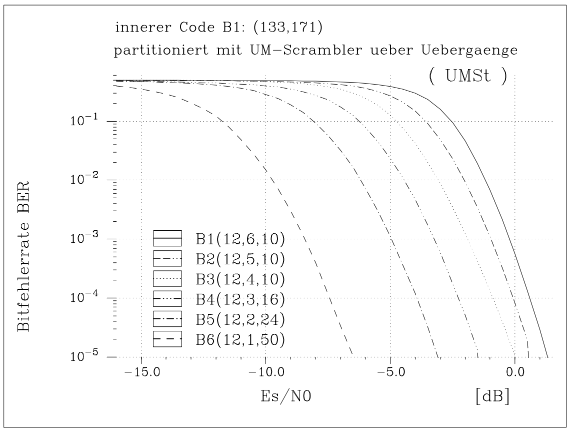


Abbildung A.2: Bitfehlerraten der Untercodes bei Partitionierung des Codes $(133, 171)$ mit UMSt

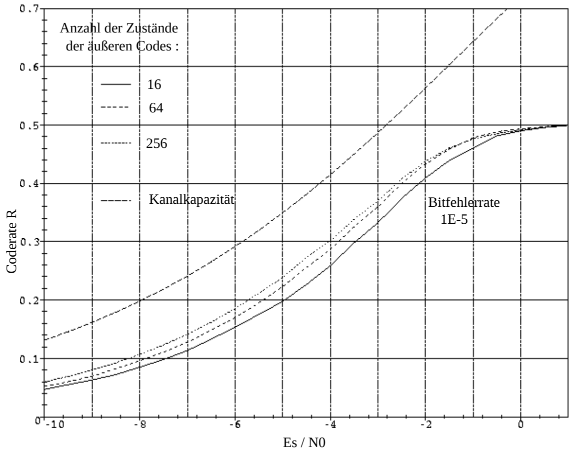


Abbildung A.3: Coderate über E_s/N_0 für GC-Codes ohne Scrambler und $\mathcal{B}^{(1)} = (133, 171)$

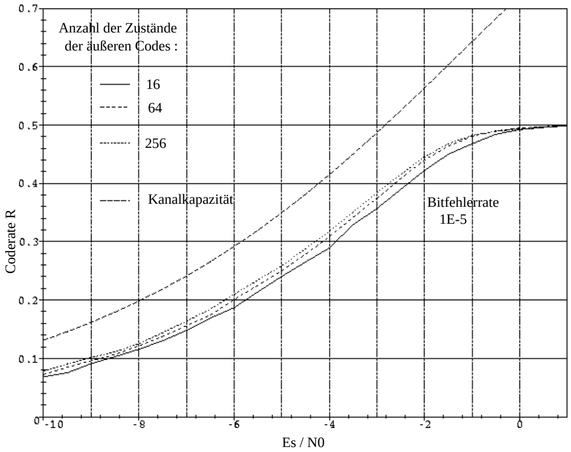


Abbildung A.4: Coderate über E_s/N_0 für GC-Codes mit UMSt und $\mathcal{B}^{(1)} = (133, 171)$

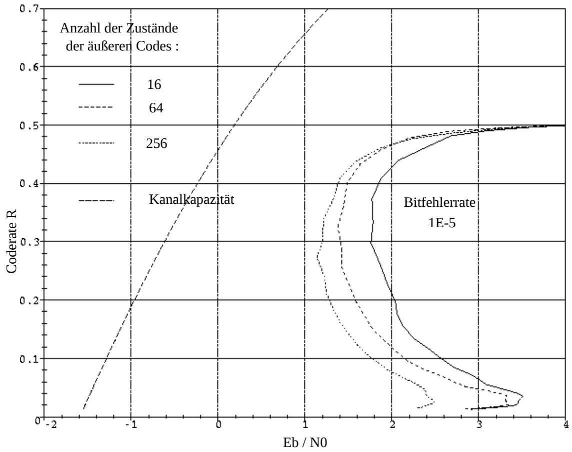


Abbildung A.5: Isoquanten für GC-Codes ohne Scrambler und $\mathcal{B}^{(1)} = (133, 171)$

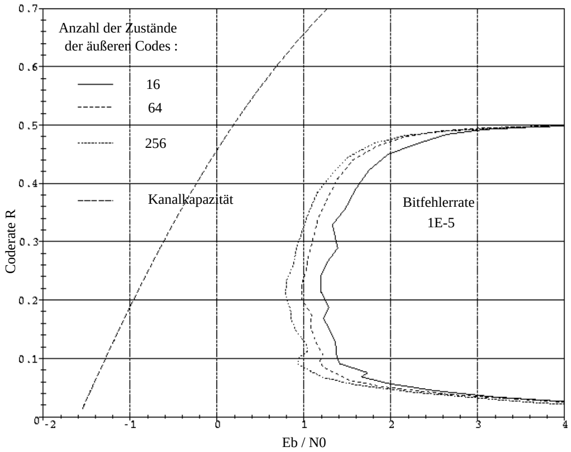


Abbildung A.6: Isoquanten für GC-Codes mit UMSt und $\mathcal{B}^{(1)} = (133, 171)$

ANHANG

B Implementierung

B.1 Scramblerkonstruktion

Der in Kapitel 3.6.3 stark vereinfacht beschriebene Algorithmus zur Scramblerkonstruktion stellt prinzipiell einen systematischen Suchalgorithmus dar. Er wurde im Rahmen dieser Arbeit zum größten Teil in `Maple` implementiert. Da `Maple` Mengenoperationen und algebraische Berechnungen sehr gut unterstützt, wurde diese Methode der ursprünglich geplanten Implementierung in der Programmiersprache `C++` vorgezogen. Um die benötigten Zeiten für die Scramblersuche dennoch möglichst kurz zu halten, wurde der zeitintensivste Teil, die Erzeugung aller Codesequenzen in Form von Zustands- oder Übergangsfolgen bis zu einem vorgegebenen Hamming-Gewicht, in der Programmiersprache `C` programmiert und in das `Maple`-Programm eingebunden. Der interne Datenaustausch zwischen beiden Programmteilen erfolgt über Dateien.

Auf die eigentliche Implementierung soll hier nicht eingegangen werden. Es erfolgt jedoch eine kurze Anleitung zur Anwendung der Hauptprogramme.

B.1.1 `Maple`-Programm zur Scramblersuche

Das Programm zur eigentlichen Scramblersuche wird von `Maple` aus aufgerufen. Dazu muß das eigentliche Programmpaket zunächst mit dem Befehl

```
read('../maple/nex6.txt');
```

geladen werden. Anschließend kann man mit

```
scramble( genpols, pct_file, scr_art, part_art, K, d_grenz, pfade, anzahl );
```

die Scramblersuche starten. Die einzelnen Parameter haben dabei folgende Bedeutung:

`genpols` : Liste der Generatorpolynome des zu partitionierenden Codes der Dimension $k = 1$ in oktaler Form (wie auf Seite 5 beschrieben).
`pct_file` Name des Files, das die Punktierungsmatrix enthält (`''` bei unpunktierten Codes)
`scr_art` : Scramblerart: BS, DS, GS (für UM-Scrambler), GS1 (ohne part. äquiv. Scr.)
`part_art` : Punktierungsart: s = Zustandspunktierung, t = Übergangspunktierung
`K` : Scramblerdimension, Anzahl der Partitionierungsstufen

`d_grenz` : 1. Abbruchkriterium:
Es werden nur Pfade bis zur dieser Hamming-Distanz bei der Scramblersuche berücksichtigt.
(ziemlich groß au wählen)

`pfade` : 2. Abbruchkriterium:
Sobald mehr als `pfade` Codefolgen vom aktuellen Gewicht d existieren, wird zu zur Suche der nächsten Spalte übergegangen.
(guter Wert: 500-1000)

`anzahl` : 3. Abbruchkriterium:
Gibt an, bei welcher Zahl von Scramblern zur Konstruktion der nächsten Spalte übergegangen wird.
0 = Automatik (Berücksichtigung der Partitionierungsäquivalenz)

Zum Beispiel wurde der UM-Scrambler zur Partitionierung durch Übergangspunktierung aus Kapitel 5 wie folgt gesucht:

```
scramble( [23,35], ‘YasudaM23_35_R4_5‘, GS , t, 4, 60, 500, 1 );
```

Das File mit der Punktierungsmatrix muß im `ASCII`-Format in folgender Form vorliegen:

```
# length :
8
# punct mask :
1 0 1 1
1 1 0 0
```

B.1.2 C-Programm zur Pfaderzeugung

Das C-Programm `save_pct_trans_cutf2.c` zur Pfaderzeugung basiert auf dem Programm `dist.c` von H. Grießer (Universität Ulm, Abteilung Informationstechnik), welches zur Berechnung der Gewichtsverteilungen von Faltungscodes dient. Es wurde um wesentliche Funktionen erweitert und ist auch eigenständig ohne das oben beschriebene `Maple`-Programm zu verwenden. Deshalb wird an dieser Stelle die Online-Hilfe des Programms aufgeführt:

```
>>>  save_pct_trans_cutf2  <<<

optional parameters:

  -k information_frame  length of information frame (default: 1)
  -n codeword_frame     length of codeword frame
                        (this value must match the number of polynoms)
  -m length_of_memory   k*m=number of memory registers without input registers
                        (default: as mutch as polynoms require)
  -l involved_blocks    k*l=number of memory registers with input registers
                        (default: as mutch as polynoms require)
  -g max_grade          max grade of distance function
                        > 0 : all paths with weight = max_grade  are printed/saved
                        < 0 : all paths with weight<=|max_grade| are printed/saved
```

```
                       (default: upper bound of free distance + 4)
  -s max_recursions    max number of recursions to prevent stack overflow
                       (default: 200)
  -r                   mirror polynoms; necessary if input of the convolutional
                       coder is at the LSB of polynoms (default: do not mirror)
                       input of convolutional coder is at the MSB of polynoms)

  -o output.file       write output as matrix in file 'output.file'
  -w                   write output as matrix in file dist.dat :
                               weight distribution
                               error weight distribution !?
                               free distance, k, n

  -f paths.file        write paths in file 'paths.file'
  -v                   view paths on display (states or transitions)
  -t                   view/save paths as transitions (default: as states)

  -c punctering.file   file which contains a punctering mask

  -q A,B,C             give states/trans. that are already deleted by a scrambler
                       A = k_scr (size of the Scrambler)
                         = 0 (k_scr is chosen automatically)
                       B = Options (a:view/save deleted OR q:don't,
                                    f:read file)
                       C = filename, if B is qf or af
                         = list,     if B is q or a

                       example 1 : 0,a,0,2,1,3,1,1,2
                       0        -> k_scr automatically (depends on -t)
                       a        -> view/save paths, mark the deleted states/trans.
                       0,2,1,3 -> in the 0. trellis-state/trans there are
                                  2 values deleted (1,3)
                       1,1,2    -> in the 1. trellis-state/trans. there is
                                  1 value deleted (2)
                       example 2 : 3,qf,test.dat
                       3        -> k_scr=3
                       qf       -> view/save all paths but the deleted
                       test.dat has to be of the form:
                                  2            (= number of following rows)
                                  0,1,3
                                  1,2          (compare to example 1)

  -d                   display more information
  -h                   show this information

necessary parameter:

  -p generator_polynoms generator polynoms of convolutional decoder
                       (decimal=nnn, oktal=0nnn, hexadecimal=0xnnn)
```

B.2 Maple-Programm zur Bestimmung der äußeren Codes

Das Programm zur Bestimmung geeigneter äußerer Codes und zur Erstellung der Isoquanten wird von `Maple` aus aufgerufen. Das Programmpaket wird dazu mit

```
read('../maple/kanalkap/ratex.txt');
```

geladen. Anschließend kann das Programm mit

```
isoquants( filename, part_stufen, rate );
```

gestartet werden. Dabei müssen folgende Parameter angegeben werden:

`filename`	: Name des Datenfiles, mit den Bitfehlerraten der inneren Codes
`part_stufen`	: Anzahl der inneren Codes bzw. Partitionierungsstufen
`rate`	: Rate des inneren Codes $\mathcal{B}^{(1)}$

Das Datenfile muß in `ASCII`-Format vorliegen und enhält in abwechselnder Reihenfolge die Werte E_s/N_0 und die zugehörige Bitfehlerwahrscheinlichkeit $\mathrm{P}_{\mathrm{B,b}}$. Durch folgendes File werden beispielsweise 2 Arbeitspunkte übergeben:

```
#         4 (R)         Code B1:
 2.51188635826111e-01   # = Es/N0
 4.98131990432739e-01   # = Bitfehlerwahrsch.
 2.81838297843933e-01
 4.99725162982941e-01
#         4 (R)         Code B2:
 2.51188635826111e-01
 4.40858781337738e-01
 2.81838297843933e-01
 4.25007432699203e-01
#         4 (R)         Code B3:
 2.51188635826111e-01
 2.94550746679306e-01
 2.81838297843933e-01
 2.55330026149750e-01
#         4 (R)         Code B4:
 2.51188635826111e-01
 2.20899209380150e-02
 2.81838297843933e-01
 1.21695213019848e-02
#        -1 (I)
```

B.3 MAP-Decoder für COSSAP

Zur Simulation der Bitfehlerkurven in Kapitel 5 wurden die beiden MAP-Algorithmen nach Abschnitt 4.2.3 als COSSAP-Module in der Programmiersprache C implementiert. Die beiden Module lauten:

MAP_ALL_1: MAP Decoder zur Decodierung des Faltungscodes $\mathcal{B}^{(1)}$ bei Verwendung eines Faltungsscramblers.

MAP_ALL_2: MAP Decoder zur Decodierung der Untercodes $\mathcal{B}^{(i)}_{\underline{z}^{(1)},\ldots,\underline{z}^{(i-1)}}$ bei verallgemeinerter Verkettung mit inneren Faltungscodes und Verwendung eines Faltungsscramblers.

Die Einstellung der Parameter für die beiden Decoder geht aus der COSSAP-Online-Hilfe bzw. dem zugehörigen .mdef-File hervor. Daher sollen an dieser Stelle nur beispielhaft zwei Input-Datasets angegeben werden, welche zum Einsatz der Decoder benötigt werden. Sie zeigen, in welcher Form die Parameter von Faltungscode und inversem Scrambler an die Module übergeben werden müssen. Dargestellt sind der Code $(23, 35)$ und der inverse UM-Scrambler (UMSt) aus Kapitel 5 bzw. Tabelle A.1:

- **Input-Dataset für einen Faltungscode:**

```
# k   = 1 = Dimension :
1
# n
2
# m + 1  = Memory + 1
5
# Oktaldarstellung des Codes nach Proakis :
23
35
```

- **Input-Dataset für einen inversen Scrambler:**

```
# Laenge l = Laenge der Partitionierungsvektoren p(i) :
5
# k_scr = Partitionierungsstufen = K :
4
# immer = 1 :
1
# Oktaldarstellungen der Partitionierungsvektoren p(0), ... p(K) :
31
27
27
20
# Verschiebungen Delta(0),...,Delta(K) im inversen Scrambler :
1
0
3
0
```

ANHANG

C

Abkürzungen und Formelzeichen

C.1 Abkürzungen

AWGN	**A**dditive **W**hite **G**aussian **N**oise
BPSK	**B**inary **P**hase **S**hift **K**eying
BS	**B**lock **S**crambler
BSs	**B**lock **S**crambler für Zustandspunktierung (**s**tates)
BSt	**B**lock **S**crambler für Übergangspunktierung (**t**ransitions)
`C`	Programmiersprache `C`
`C++`	Programmiersprache `C++`
CC	**C**oncatenated **C**ode
CCF	**C**ontroller **C**anonical **F**orm
CS	**C**onvolutional **S**crambler (Faltungsscrambler)
CSs	CS für Zustandspunktierung (**s**tates)
CSt	CS für Übergangspunktierung (**t**ransitions)
`COSSAP`	**C**ommunication **S**ystem **S**imulation and **A**nalysis **P**ackage
DS	**D**iagonal **S**crambler (Faltungsscrambler in Diagonalform)
DSs	**D**iagonal **S**crambler für Zustandspunktierung (**s**tates)
DSt	**D**iagonal **S**crambler für Übergangspunktierung (**t**ransitions)
FIR	**F**inite **I**mpulse **R**esponse
GC	**G**eneralized **C**oncatenation (Verallgemeinerte Codeverkettung)
GCC	**G**eneralized **C**oncatenated **C**ode
HD	**H**ard–**D**ecision
IIR	**I**nfinite **I**mpulse **R**esponse
`Maple`	**Maple** Computer Algebra System
ML	**M**aximum–**L**ikelihood
MAP	**M**aximum **A** **P**osteriori
SD	**S**oft–**D**ecision
UM	**U**nit **M**emory
UMS	**U**nit **M**emory **S**crambler
UMSs	**U**nit **M**emory **S**crambler für Zustandspunktierung (**s**tates)
UMSt	**U**nit **M**emory **S**crambler für Übergangspunktierung (**t**ransitions)

C.2 Formelzeichen

$\underline{0}$	Nullmatrix oder Nullvektor (je nach Zusammenhang)
$\underline{0}_{[k]}$	$(k \times k)$ - Nullmatrix
$\underline{a}$	Hilfsvariable
a_{grenz}	maximale Anzahl gleichwertiger Spalten, die der Algorithmus zur Scramblersuche liefern soll
$\underline{b}$	Hilfsvariable
$b_{\text{S}}^{(i)}$	Spaltenzahl des i-ten Blockinterleavers
$b_{\text{Z}}^{(i)}$	Zeilenzahl des i-ten Blockinterleavers
$\mathcal{B}^{(1)}$	innerer Faltungscode bei verallgemeinerter Codeverkettung
$\mathcal{B}^{(i)}$	linearer Untercode von $\mathcal{B}^{(1)}$
$\mathcal{B}^{(i)}_{\underline{z}^{(1)},\ldots}$	Untercode von $\mathcal{B}^{(1)}$
c_l	l-tes Bit der Codebitfolge
$\underline{c}$	Codebitfolge
$\underline{c}_{\text{pkt}}$	Codebitfolge eines punktierten Codes
$\mathcal{C}$	(Faltungs-) Code
$\mathcal{C}_{UM}$	Unit Memory Code
d	freie Distanz eines Faltungscodes
$d^{(i)}$	Freie Distanz der i-ten Partitionierungsstufe
$d_z^{(i)}$	Freie Distanz des z-ten Untercodes der i-ten Partitionierungsstufe
d_{grenz}	maximale Hamming-Distanz, die für die Scramblersuche berücksichtigt werden soll
E_b	Energie pro Informationsbit
E_s	Energie pro Sendesymbol
$\mathbf{E}_d^{(i)}$	Menge der Code-Übergangsfolgen $\underline{\gamma}$ des Untercodes $\mathcal{B}^{(i)}$ vom Gewicht d
$g^{(j)}$	Oktales „Generatorpolynom“
$g_l^{(i,j)}$	Koeffizient der Untermatrix $\underline{G}_l$
$g^{(i,j)}(D)$	Generatorpolynom
$\underline{G}$	Generatormatrix in halbunendlicher Form
$\underline{G}(D)$	Generatormatrix in Polynomschreibweise
$\underline{G}_l$	Untermatrix der Generatormatrix
$\underline{H}$	Verschiebematrix zur Berechnung partitionierungsäquivalenter Scrambler
$\underline{H}(D)$	polynomiale Verschiebematrix
$\underline{H}^{\delta}$	δ-te Potenz von $\underline{H}$
i	Laufvariable
i_l	$l-$tes Bit der Informationsfolge $\underline{i}$
$\underline{i}$	Informationsbitfolge
$\widehat{\underline{i}}^{(j)}$	Decodierte Informationsfolge des äußeren Codes $\mathcal{A}^{(i)}$
$\underline{I}_{[k]}$	$(k \times k)$-Einheitsmatrix
j	Laufvariable
k	Dimension eines Faltungscodes
K	Dimension eines UM-Codes
$K^{(i)}$	Dimension der Untercodes der i-ten Partitionierungsstufe
$\mathbf{K}$	Menge aller möglichen $(\underline{p}, \Delta)$-Kombinationen
$\mathbf{K}_1$	Menge aller $(\underline{p}, 1)$-Kombinationen
$\mathbf{K}_{\text{best}}$	Menge von $(\underline{p}, \Delta)$-Kombinationen
$\lvert\mathbf{K}_{\text{best}}\rvert$	Mächtigkeit der Menge $\mathbf{K}_{\text{best}}$
$\widehat{\mathbf{K}}^{(i)}$	Menge von Matrixspalten $\underline{P}^{(i)}$

l	Laufvariable ; Länge eines Partitionierungsvektors $\underline{p}^{(i)}$
l_{Info}	Blocklänge eines terminierten Faltungscodes (in Informationsbits)
l_{Code}	Blocklänge eines terminierten Faltungscodes (in Codebits)
l_{Term}	Terminierungslänge eines Faltungscodes (in Informationsbits)
L	Blocklänge eines terminierten Faltungscodes (in Übergängen)
$\underline{m}, \underline{m}', \underline{m}''$	Trelliszustände im MAP-Algorithmus nach Bahl
m	Gesamteinflußlänge eines Faltungscodes
m_{scr}	Gesamteinflußlänge eines Scramblers
m_{inv}	Gesamteinflußlänge eines inversen Scramblers
M	Gesamteinflußlänge eines UM-Codes oder Scramblers
$\underline{M}$	Scramblermatrix in halbunendlicher Form
$\underline{M}(D)$	Scramblermatrix in Polynomialdarstellung
M'	Gesamteinflußlänge eines äquivalenten Faltungscodes
$\mathbf{M}^{(i)}$	Menge möglicher Teilmatrizen $[\underline{P}^{(1)}, \ldots, \underline{P}^{(i)}]$
n	Anzahl der Elemente eines Codevektors $\underline{v}_t$
N	Anzahl der Elemente eines Codevektors $\underline{y}_t$ bei UM-Codes
N_0	einseitige Rauschleistungsdichte
p_{pkt}	Punktierungsperiode
$\underline{p}^{(i)}$	Partitionierungsvektor (Teil des i-ten Spaltenvektors von $\underline{M}^{-1}$
$p_l^{(i)}$	l-tes Element des Vektors $\underline{p}^{(i)}$
$(p^{(i)}, \Delta^{(i)})$	spezielle Darstellung einer inversen Scramblermatrix
$\underline{P}$	Teilmatrix der inversen Scramblermatrix $\underline{M}^{-1}$
$\underline{P}_0$	untere Teilmatrix von $\underline{P}$
$\underline{P}_1$	obere Teilmatrix von $\underline{P}$
$\underline{P}_{\text{pkt}}$	Punktierungsmatrix
$\mathbf{P}^{(i)}$	Menge von Vektoren $\underline{p}^{(i)}$
$\mathbf{P}$	Menge aller möglichen 2^l binären Partitionierungsvektoren der Länge l
P	Wahrscheinlichkeit
$\mathrm{P_b}$	Bitfehlerrate
$q^{(i)}(D)$	i-tes Rückkoppelpolynom eines rekursiven Faltungscodes
r	Coderate eines Faltungscodes
r_{pkt}	Coderate eines punktierten Faltungscodes
R	Coderate eines UM-Codes
$\underline{u}$	Informationsfolge eines Faltungscodes
$\underline{u}_t$	Eingangsvektor eines Faltungscodes zum Zeitpunkt t mit k Informationsbits
u_t	Eingangsbit zum Zeitpunkt t bei einem Faltungscode der Dimension $k = 1$
$u_t^{(i)}$	i-tes Element von $\underline{u}_t$
$\underline{u}^{(i)}, u^{(i)}(D)$	Folge der i-ten Informationsbits $u_t^{(i)}$
$\underline{u}(D)$	Informationsfolge in Polynomialschreibweise
$\underline{v}$	Codebitfolge eines Faltungscodes
$\underline{v}_t$	Codevektor aus n Bits
$v_t^{(j)}$	j-tes Element von $\underline{v}_t$
$\underline{v}^{(j)}, v^{(j)}(D)$	Folge der j-ten Codebits $u_t^{(j)}$
$\underline{v}(D)$	Codefolge in Polynomialschreibweise
$\underline{\tilde{v}}$	Empfangsfolge
$\mathbf{V}$	Menge aller möglicher Zustände oder Übergänge im Trellisdiagramm
$\mathbf{V}_\Delta^{(i)}$	Menge nicht punktierter Zustände zu den Zeitpunkten mit Verschiebung Δ

$\overline{\mathbf{V}}_{\Delta}^{(i)}$	Menge punktierter Zustände zu den Zeitpunkten mit Verschiebung Δ
$\underline{x}$	Informationsfolge eines UM-Codes
$\underline{x}_{\tau}$	Eingangsvektor eines UM-Codes zum Zeitpunkt τ
$x_{\tau}^{(i)}$	i-tes Element von $\underline{x}_{\tau}$
$\underline{x}^{(i)}, x^{(i)}(D)$	Folge der i-ten Informationsbits $x_{\tau}^{(i)}$ eines UM-Codes
$\underline{x}(D)$	Informationsfolge eines UM-Codes in Polynomialschreibweise
$\overline{\mathbf{X}}_{\Delta}^{(i)}$	Menge von Zuständen (Übergängen), die in $t = t_0 - \Delta$ punktiert werden sollen
$\underline{y}$	Codefolge eines UM-Codes
$\underline{y}_{\tau}$	Codevektor eines UM-Codes
$\underline{y}_{\tau}^{(j)}$	j-tes Element von $\underline{y}_{\tau}$
$\underline{y}^{(j)}, y^{(i)}(D)$	Folge der j-ten Codebits eines UM-Codes
$\underline{y}(D)$	Codefolge eines UM-Codes in Polynomialschreibweise
$\underline{y}_{\underline{z}^{(1)},\ldots}$	Codefolge eines Untercodes
$\underline{\tilde{y}}$	Empfangsfolge in UM-Darstellung
z_{pkt}	Anzahl von Einsen in einer Punktierungsmatrix
$\underline{z}$	Informationsfolge am Scramblereingang
$\underline{z}_{\tau}$	Eingangsvektor eines Scramblers τ
$z_{\tau}^{(i)}$	i-tes Element von $\underline{z}_{\tau}$
$\underline{z}^{(i)}, x^{(i)}(D)$	Folge der i-ten Informationsbits $z_{\tau}^{(i)}$ am Scramblereingang
$\underline{z}(D)$	Codefolge am Scramblereingang in Polynomialschreibweise
$\underline{\tilde{z}}_{\tau}^{(i)}$	Wahrscheinlichkeit für $\underline{z}_{\tau}^{(i)} = 1$
$\underline{\tilde{z}}^{(i)}$	Wahrscheinlichkeitsfolge am Decoderausgang
$\underline{\widehat{x}}_{\tau}^{(i)}$	Decodiertes Codebit des äußeren Codes $\mathcal{A}^{(i)}$
$\underline{\widehat{x}}^{(i)}$	Folge decodierter Codebits des äußeren Codes $\mathcal{A}^{(i)}$
$\alpha_t, \beta_t, \sigma_t, \xi_t$	Wahrscheinlichkeitswerte im MAP-Algorithmus nach Bahl
$\underline{\gamma}$	Übergangsfolge im Trellis
$\underline{\gamma}_t, \gamma_t$	Übergang im Trellis zum Zeitpunkt t
$\underline{\Gamma}$	Übergangsfolge im UM-Trellis
$\underline{\Gamma}_{\tau}$	Übergang im UM-Trellis zum Zeitpunkt τ
δ	Offset von Δ bei einem partitionierungsäquivalenten Scrambler
Δ	Entfernung zum nächsten UM-Zustand im konventionellen Trellis
$\Delta^{(i)}$	Verschiebung des Vektors $\underline{p}^{(i)}$ in der i-ten Spalte von $\underline{M}^{-1}$
$\Delta E_{s,[\mathrm{dB}]}$	(zusätzlicher) Codierungsgewinn in dB
Λ	$= \{0, 1, \ldots, K-1\}$, Menge aller zulässigen Verschiebungen Δ
$\underline{\theta}$	Zustandsfolge im Trellis
$\underline{\theta}_t, \theta_t$	Zustand im Trellis zum Zeitpunkt t
$\underline{\Theta}$	Zustandsfolge im UM-Trellis
$\underline{\Theta}_{\tau}$	Zustand im UM-Trellis zum Zeitpunkt τ
ν	Gesamteinflußlänge (overall constraint length)
ν_i	Einflußlänge (constraint length)
$\mathcal{V}$	Gesamteinflußlänge bei UM-Codes
$\mathcal{V}_i$	Einflußlänge bei UM-Codes

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[And94] J.D. Andersen. *The TURBO Coding Scheme, Report IT-146 ISSN 0105-854.* Technical report, Technical University of Denmark, Institute of Telecommunication, (1994).

[BCJR74] L.R. Bahl, J. Cocke, F. Jelink und J. Raviv. *Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate.* IEEE Transactions on Information Theory **IT-20** (Mar. 1974), pp. 284–287.

[BDS96a] M. Bossert, H. Dieterich und S.A. Shavgulidze. *Generalized Concatenation of Convolutional Codes.* European Transactions on Telecommunication **Vol. 7** (Nov./Dec. 1996) No. 6, pp. 483–492.

[BDS96b] M. Bossert, H. Dieterich und S.A. Shavgulidze. *Partitioning of Convolutional Codes using a Convolutional Scrambler.* Electronic Letters **Vol. 32** (September 1996) No. 19, pp. 1758–1760.

[BM94] Nancy R. Blachman und Michael J. Mossinghoff. *Maple V Quick Reference.* Brooks/Cole Publishing Company, Pacific Grove, California (1994).

[Bos92] M. Bossert. *Kanalcodierung.* B.G. Teubner, Stuttgart (1992).

[Bos94] M. Bossert. *Lineare Passive Systeme, Signale und Systeme.* Skript und Hilfsblätter zur Vorlesung, 1994.

[BZ74] E.L. Blokh und V.V. Zyablov. *Coding of Generalized Cascade Codes.* Problemy Peredachi Informatsii **Vol. 10** (1974), pp. 45–50.

[Die96] Hans Dieterich. *Partitioning of Binary Convolutional Codes.* Vorabdruck, 1996.

[DSV94] Uwe Dettmar, Ulrich K. Sorger und Zyablov Victor V. *Concatenated Codes with Convolutional Inner and Outer Codes.* Lecture Notes in Computer Science **829** (1994).

[For70] G.D. Forney. *Convolutional Codes I: Algebraic Structure.* IEEE Transactions on Information Theory **IT-16** (Nov. 1970), pp. 720–738.

[Fü96] Uwe Füssel. *Iterative Decodierung von Codes basierend auf Faltungscodes und MSK Modulation.* Diplomarbeit, Universität Ulm, Abt. Informationstechnik, 1996.

[Haa96] Günther Haas. *Negazyklische Codes und Lee-Metrik.* Studienarbeit, Universität Ulm, Abt. Informationstechnik, 1996. (S/1995/JM2).

[HJZ] Stefan Höst, Rolf Johannesson und Viktor V. Zyablov. *On the Construction of Concatenated Codes Based on Binary Conventional Convolutional Codes.*

[Hol88] K.J. Hole. *New Short Constraint Length Rate (N-1)/N Punctured Convolutional Codes for Soft-Decision Viterbi Decoding.* IEEE Transactions on Information Theory **IT-34** (Sept. 1988), pp. 1079–1081.

[JTZ88] Jørn Justesen, Christian Thommesen und Victor V. Zyablov. *Concatenated Codes with Convolutional Inner Codes.* IEEE Transactions on Information Theory **Vol. 34** (September 1988) No. 5, pp. 1217–1225.

[JW93] R. Johannesson und Zhe-xian Wan. *A Linear Algebra Approach to Minimal Convolutional Encoders.* IEEE Transactions on Information Theory **IT-39** (July 1993), 1219–1233.

[JZZ95] R. Johannesson, K.Sh. Zigangirov und Viktor Zyablov. *Lower Bounds on the Free Distance for Random Concatenated Convolutional Codes.* (1995). preprint.

[KR83] B.W. Kernighan und D.M. Ritchie. *Programmieren in C.* Janser, München (1983).

[Kre89] Ulrich H.-G. Kreßel. *Informationstheoretische Beurteilung digitaler Übertragungsverfahren mit Hilfe des Fehlerexponenten*, Fortschr.-Ber. VDI Reihe 10Bd. Nr. 121. VDI Verlag, Düsseldorf (1989).

[Lar73] K.J. Larsen. *Short Convolutional Codes with Maximal Free Distance for Rates 1/2, 1/3 and 1/4.* IEEE Transactions on Information Theory **IT-19** (May 1973), 371–372.

[Lee76] Lin-Nan Lee. *Short Unit-Memory Byte-Oriented Binary Convolutional Codes having Maximal Free Distance.* IEEE Transactions on Information Theory **IT-22** (May 1976), pp. 349–352.

[Lin93] J. Lindner. *Nachrichtentechnik I,II.* Vorlesungsmitschrift, 1993.

[Mau94] Johannes Maucher. *Bounded Minimum Distance Decodierung für (Partial)Unit-Memory-Codes.* Diplomarbeit, Universtät Ulm, Abt. Informationstechnik, 1994.

[Paa74] Erik Paaske. *Short Binary Convolutional Codes with Maximal Free Distance for Rates 2/3 and 3/4.* IEEE Transactions on Information Theory (1974).

[Pal95] Reginaldo Palazzo. *A Network Flow Approach to Convolutional Codes.* IEEE Transactions on Communications **Vol. 43** (February/March/April 1995) No. 2/3/4, pp. 1429–1440.

[Pro89] John G. Proakis. *Digital Communications*, 2. Aufl. Computer Science. McGRAW-HILL International Editions, (1989).

[RRZ93] RRZN. *Die Programmiersprache C. Ein Nachschlagewerk.* Universität Hannover, (1993).

[Sch95] Walter Schnug. *Kombination von Kanalcodierung und Kanalschätzung bei Mehrfrequenzübertragung.* Diplomarbeit, Universtät Ulm, Abt. Informationstechnik, 1995.

[TJ83] Christian Thommesen und Jørn Justesen. *Bounds on Distances and Error Exponents of Unit memory Codes.* IEEE Transactions on Information Theory **IT-29** (September 1983) 5, pp. 637–649.

[VO79] Andrew J. Viterni und Jim K. Omura. *Principles of Digital Communication and Coding.* McGraw-Hill Book Company, New York (1979).

[WH93] Thomas Woerz und Joachim Hagenauer. *Decoding of M-PSK-Multilevel Codes.* European Transactions on Telecommunication **Vol. 4** (May-June 1993) No. 3, pp. 299–308.

[WHPH87] William W. Wu, David Haccoun, Robert Peile und Yasuo Hirata. *Coding for Satellite Communication.* IEEE Journal on Selected Areas in Communications **SAC-5** (May 1987) No. 4, 737ff.

[YKH84] Yutaka Yasuda, Kanshiro Kashiki und Yasuo Hirata. *High-Rate Punctured Convolutional Codes for Soft decision Viterbi Decoding.* IEEE Transactions on Communications **COM-32** (March 1984) No. 3, pp. 315–319.

[ZS91] V.V. Zyablov und Shavgulidze S.A. *Generalized Concatenated Constructions on the Base of Convolutional Codes.* Nauka, (1991).