Überblick zu Kapitel 4 von „Einführung in die Kanalcodierung"

Das letzte Kapitel dieses Buches beschreibt **iterative Decodierverfahren,** wie sie in den meisten heutigen (2016) Kommunikationssystemen eingesetzt werden. Dies hat folgende Gründe:

- Um sich der Kanalkapazität anzunähern, benötigt man sehr lange Codes.
- Für lange Codes ist aber eine blockweise Maximum–Likelihood–Decodierung nahezu unmöglich.

Die Decoder–Komplexität lässt sich bei nahezu gleicher Qualität deutlich herabsetzen, wenn man zwei (oder mehrere) kurze Kanalcodes miteinander verknüpft und beim Empfänger die jeweils neu gewonnene (Soft–)Information in mehreren Schritten – also iterativ – zwischen den Decodern austauscht.

Der Durchbruch auf dem Gebiet gelang Anfang der 1990er Jahre mit der Erfindung der **Turbo–Codes** durch **Claude Berrou** und kurz darauf mit der Wiederentdeckung der **Low–density Parity–check Codes** durch D.J.C. MacKay und R.M. Neal, nachdem die schon 1961 von **Robert G. Gallager** entwickelten LDPC–Codes zwischenzeitlich in Vergessenheit geraten waren.

Im Einzelnen werden in diesem Kapitel behandelt:

- Eine Gegenüberstellung von *Hard Decision* und *Soft Decision*,
- die Quantifizierung von *Zuverlässigkeitsinformation* durch *Log Likelihood Ratios*,
- das Prinzip der *symbolweisen Soft–in Soft–out (SISO) Decodierung*,
- die Definition von *Apriori–*, *Aposteriori–* und *extrinsischem L–*Wert,
- die Grundstruktur von *seriell bzw. parallel verketteten* Codiersystemen,
- die Eigenschaften von *Produkt–Codes* und deren *Hard Decision Decodierung*,
- die Grundstruktur, der Decodieralgorithmus und die Leistungsfähigkeit der *Turbo–Codes*,
- Grundlegendes zu den *Low–density Parity–check Codes* und deren Anwendungen.

Die grundlegende Theorie wird auf 32 Seiten dargelegt. Dazu beinhaltet das Kapitel noch 94 Grafiken, 13 Aufgaben und 8 Zusatzaufgaben mit insgesamt 102 Teilaufgaben, sowie ein Lernvideo (LV) und zwei Interaktionsmodule (IM):

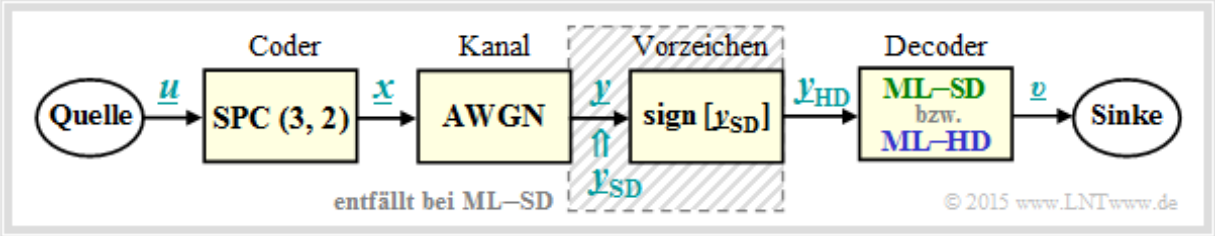
- **Galoisfeld: Eigenschaften und Anwendungen** (LV zu den Grundlagen,Gesamtdauer 39:10)
- **Komplementäre Gaußsche Fehlerfunktionen** (IM zu den Grundlagen)
- **Viterbi–Algorithmus** (IM zu den Kapitel 3.4 und 4.3)

Geeignete Literatur: [BCJR74] – [BGT93] – [Big05] – [Bos98] – [Bos99] – [CT06] – [Eli54] – [Eli55] – [Gal63] – [Hag90] – [JKE00] – [KM+08] – [Liv10] – [Liv15] – [McE96] – [MN97] – [RL09] – [RU08] – [Schr07] – [Sha48] – [Vit67]

Einige Grafiken basieren auf den Daten aus [Bos98], [Liv15], **[Böh14]** und **[Str14].** Wir danken den Autoren für die Nutzungserlaubnis und Dr.–Ing. **Ronald Böhnke** für die Korrekturen zu diesem Kapitel.

Hard Decision vs. Soft Decision (1)

Zur Hinleitung auf die Thematik dieses vierten und letzten Kapitels und zur Motivation betrachten wir das folgende Nachrichtenübertragungssystem mit Codierung.



Nachfolgend werden alle Symbole in bipolarer Darstellung angegeben: „0” → „+1” und „1” → „–1”.

- Die Symbolfolge $\underline{u} = (u_1, u_2)$ wird der Coderfolge $\underline{x} = (x_1, x_2, x_3) = (u_1, u_2, p)$ zugeordnet, wobei für das Paritybit $p = u_1 \oplus u_2$ gilt ⇒ **Single Parity–check Code** ⇒ SPC (3, 2, 2).
- Der **AWGN–Kanal** verändert die Binärsymbole $x_i \in \{+1, -1\}$ zu reellwertigen Ausgangswerten y_i, z.B. im Block 4 der unteren Tabelle: $x_1 = +1 \Rightarrow y_1 = +0.9$ und $x_3 = -1 \Rightarrow y_3 = +0.1$.
- Die Decodierung geschieht gemäß dem Kriterium **Maximum Likelihood (block–wise ML),** wobei zwischen *Hard Decision* (ML–HD) und *Soft Decision* (ML–SD) zu unterscheiden ist.
- Das obige Blockschaltbild in seiner Gesamtheit entspricht ML–HD. Hier werden zur Detektion nur die Vorzeichen der AWGN–Ausgangswerte entsprechend $y_{\text{HD},\,i} = \text{sign}[y_{\text{SD},\,i}]$ ausgewertet.
- Bei *Soft Decision* (ML–SD) verzichtet man auf den schraffierten Block in obigem Blockschaltbild und wertet direkt die wertkontinuierlichen Eingangsgrößen $y_{\text{SD},\,i}$ aus.

	Block 1	Block 2	Block 3	Block 4
$\underline{u}$	+1, –1	+1, –1	+1, –1	+1, –1
$\underline{x}$	+1, –1, –1	+1, –1, –1	+1, –1, –1	+1, –1, –1
$\underline{y}_{\text{SD}}$	+1.0, –1.0, –1.0	+0.9, –0.1, –0.8	+0.9, +0.1, –0.8	+0.9, +0.1, +0.1
$\underline{v}_{\text{SD}}$	+1, –1	+1, –1	+1, –1	+1, +1 Fehler
$\underline{y}_{\text{HD}}$	+1, –1, –1	+1, –1, –1	+1, +1, –1	+1, +1, +1
$\underline{v}_{\text{HD}}$	+1, –1	+1, –1	E, E Erasures	+1, +1 Fehler

gültig für Soft Decision

gültig für Hard Decision

©2015 www.LNTwww.de

Aus der Beispieltabelle erkennt man:

- **Hard Decision** ⇒ die Symbolfolge $\underline{v}_{\text{HD}}$ ergibt sich aus den hart entschiedenen Kanalwerten $\underline{y}_{\text{HD}}$ (blaue Hinterlegung): Es werden nur die beiden ersten Blöcke fehlerfrei decodiert.
- **Soft Decision** ⇒ die Symbolfolge $\underline{v}_{\text{SD}}$ ergibt sich aus den „weichen” Kanalausgangswerten $\underline{y}_{\text{SD}}$ (grüne Hinterlegung): Nun wird in diesem Beispiel auch der dritte Block richtig entschieden.

Die Bildbeschreibung wird auf der nächsten Seite fortgesetzt.

Hard Decision vs. Soft Decision (2)

Für alle Spalten der folgenden Tabelle wird vorausgesetzt:

- der Nachrichtenblock $\underline{u} = (0, 1)$, bipolar darstellbar als $(+1, –1)$,
- der SPC (3, 2)–codierte Block $\underline{x} = (0, 1, 1)$ bzw. in Bipolardarstellung $(+1, –1, –1)$.

Die vier Blöcke unterscheiden sich also nur durch unterschiedliche AWGN–Realisierungen.

	Block 1	Block 2	Block 3	Block 4
$\underline{u}$	+1, –1	+1, –1	+1, –1	+1, –1
$\underline{x}$	+1, –1, –1	+1, –1, –1	+1, –1, –1	+1, –1, –1
$\underline{y}_{SD}$	+1.0, –1.0, –1.0	+0.9, –0.1, –0.8	+0.9, +0.1, –0.8	+0.9, +0.1, +0.1
$\underline{v}_{SD}$	+1, –1	+1, –1	+1, –1	+1, +1 Fehler
$\underline{y}_{HD}$	+1, –1, –1	+1, –1, –1	+1, +1, –1	+1, +1, +1
$\underline{v}_{HD}$	+1, –1	+1, –1	E, E Erasures	+1, +1 Fehler

gültig für Soft Decision

gültig für Hard Decision

Die Tabelle ist wie folgt zu interpretieren:

- Bei idealem Kanal entsprechend Block 1 $\Rightarrow$ $\underline{x} = \underline{y}_{SD} = \underline{y}_{HD}$ gibt es keinen Unterschied zwischen der (blauen) herkömmlichen ML–HD –Variante und der (grünen) ML–SD –Variante.
- Der Block 2 demonstriert geringe Signalverfälschungen. Wegen $\underline{y}_{HD} = \underline{x}$ (das heißt, dass der Kanal die Vorzeichen nicht verfälscht) liefert auch ML–HD das richtige Ergebnis $\underline{v}_{HD} = \underline{u}$.
- Dagegen gilt im dritten Block $\underline{y}_{HD} \neq \underline{x}$ und es gibt auch keine SPC (3, 2)–Zuordnung $\underline{u} \Rightarrow \underline{y}_{HD}$. Der ML–Decoder kann hier nur durch die Ausgabe $\underline{v}_{HD} = (E, E)$ vermelden, dass er bei der Decodierung dieses Blocks gescheitert ist. „E" steht hierbei für **Erasure** (deutsch: *Auslöschung*).
- Auch der *Soft Decision* Decoder erkennt, dass eine Decodierung anhand der Vorzeichen nicht funktioniert. Anhand der $\underline{y}_{SD}$–Werte erkennt er aber, dass mit großer Wahrscheinlichkeit das zweite Bit verfälscht wurde und entscheidet sich für die richtige Symbolfolge $\underline{v}_{SD} = (+1, –1) = \underline{u}$.
- Im vierten Block werden durch den AWGN–Kanal sowohl die Vorzeichen von Bit 2 als auch von Bit 3 verändert, was zum Ergebnis $\underline{v}_{HD} = (+1, +1) \neq \underline{u}\ (+1, –1)$ führt $\Rightarrow$ ein Blockfehler und gleichzeitig ein Bitfehler. Auch der *Soft Decision* Decoder liefert hier das gleiche falsche Ergebnis.

Die Decodiervariante ML–SD bietet gegenüber ML–HD zudem den Vorteil, dass man relativ einfach jedes Decodierergebnis mit einem Zuverlässigkeitswert versehen kann (in obiger Tabelle ist dieser allerdings nicht angegeben). Dieser Zuverlässigkeitswert

- hätte bei Block 1 seinen Maximalwert,
- wäre bei Block 2 deutlich kleiner,
- läge bei Block 3 und 4 nahe bei 0.

Zuverlässigkeitsinformation – Log Likelihood Ratio (1)

Es sei $x \in \{+1, -1\}$ eine binäre Zufallsvariable mit den Wahrscheinlichkeiten $\Pr(x = +1)$ und $\Pr(x = -1)$. Für die Codierungstheorie erweist es sich als zweckmäßig hinsichtlich der Rechenzeiten, wenn man anstelle der Wahrscheinlichkeiten $\Pr(x = \pm 1)$ den natürlichen Logarithmus des Quotienten heranzieht.

Definition: Das **Log–Likelihood–Verhältnis** (kurz: der L–Wert, englisch: *Log–Likelihood Ratio*, LLR) der Zufallsgröße $x \in \{+1, -1\}$ lautet:

$$L(x) = \ln \frac{\Pr(x = +1)}{\Pr(x = -1)}.$$

Bei unipolarer/symbolhafter Darstellung ($+1 \rightarrow 0$ und $-1 \rightarrow 1$) gilt entsprechend mit $\xi \in \{0, 1\}$:

$$L(\xi) = \ln \frac{\Pr(\xi = 0)}{\Pr(\xi = 1)}.$$

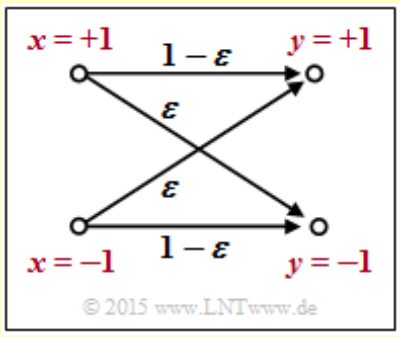
Nachfolgend ist der nichtlineare Zusammenhang zwischen $\Pr(x = \pm 1)$ und $L(x)$ angegeben. Ersetzt man $\Pr(x = +1)$ durch $\Pr(\xi = 0)$, so gibt die mittlere Zeile den L–Wert der unipolaren Zufallsgröße ξ an.

$\Pr(x = +1)$	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.0
$L(x)$	$+\infty$	+2.197	+1.382	+0.847	+0.405	0	–0.405	–0.847	–1.382	–2.197	$-\infty$
$\Pr(x = -1)$	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Man erkennt:

- Der wahrscheinlichere Zufallswert von $x \in \{+1, -1\}$ ist durch $\operatorname{sign} L(x)$ $\Rightarrow$ **Vorzeichen** gegeben.
- Dagegen gibt der **Betrag** $|L(x)|$ die Zuverlässigkeit für das Ergebnis $\operatorname{sign}(L(x))$ an.

Beispiel: Wir betrachten das skizzierte **BSC–Modell** mit bipolarer Darstellung. Hier gilt mit der Verfälschungswahrscheinlichkeit $\varepsilon = 0.1$ sowie den beiden Zufallsgrößen $x \in \{+1, -1\}$ und $y \in \{+1, -1\}$ am Eingang und Ausgang des Kanals:



$$L(y\,|\,x) = \ln \frac{\Pr(y\,|\,x = +1)}{\Pr(y\,|\,x = -1)} = \begin{cases} \ln\,[(1-\varepsilon)/\varepsilon] & \text{für } y = +1, \\ \ln\,[\varepsilon/(1-\varepsilon)] & \text{für } y = -1. \end{cases}$$

Beispielsweise ergeben sich für $\varepsilon = 0.1$ folgende Zahlenwerte (vergleiche obere Tabelle):

$$L(y = +1\,|\,x) = \ln \frac{0.9}{0.1} = +2.197, \quad L(y = -1\,|\,x) = -2.197.$$

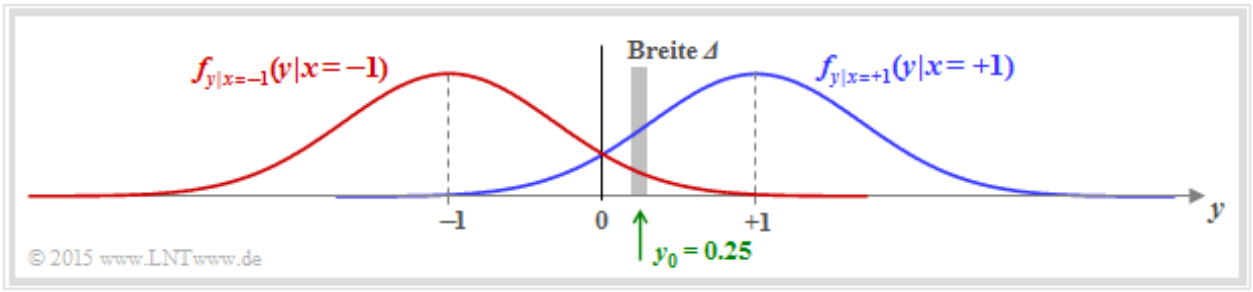
Dieses Beispiel zeigt, dass man die sog. L–Wert–Algebra auch auf bedingte Wahrscheinlichkeiten anwenden kann. In **Aufgabe Z4.1** wird das BEC–Modell in ähnlicher Weise beschrieben.

Zuverlässigkeitsinformation – Log Likelihood Ratio (2)

Wir betrachten nun den AWGN–Kanal mit den beiden bedingten Wahrscheinlichkeitsdichtefunktionen

$$f_{y|x=+1}(y\,|\,x=+1) \;=\; \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot \mathrm{e}^{-(y-1)^2/(2\sigma^2)},$$
$$f_{y|x=-1}(y\,|\,x=-1) \;=\; \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot \mathrm{e}^{-(y+1)^2/(2\sigma^2)}.$$

In der Grafik sind zwei beispielhafte Gaußfunktionen als blaue bzw. rote Kurve dargestellt.



Die gesamte WDF des Ausgangssignals y ergibt sich aus der (gleich) gewichteten Summe:

$$f_y(y) = 1/2\cdot\left[f_{y|x=+1}(y\,|\,x=+1) \;+\; f_{y|x=-1}(y\,|\,x=-1)\right].$$

Berechnen wir nun die Wahrscheinlichkeit, dass der Empfangswert y in einem (sehr) schmalen Intervall der Breite Δ um $y_0 = 0.25$ liegt, so erhält man näherungsweise

$$\Pr(|y-y_0|\le \Delta/2\,\big|\,x=+1) \;\approx\; \frac{\Delta}{\sqrt{2\pi}\cdot\sigma}\cdot \mathrm{e}^{-(y_0-1)^2/(2\sigma^2)},$$
$$\Pr(|y-y_0|\le \Delta/2\,\big|\,x=-1) \;\approx\; \frac{\Delta}{\sqrt{2\pi}\cdot\sigma}\cdot \mathrm{e}^{-(y_0+1)^2/(2\sigma^2)}.$$

Die etwas größeren senkrechten Striche bezeichnen die Bedingungen, die kleineren die Betragsbildung.

Der L–Wert der bedingten Wahrscheinlichkeit in Vorwärtsrichtung (das bedeutet: Ausgang y für einen gegebenen Eingang x) ergibt sich somit als der Logarithmus des Quotienten beider Ausdrücke:

$$L(y=y_0\,|\,x) \;=\; \ln\left[\frac{\mathrm{e}^{-(y_0-1)^2/(2\sigma^2)}}{\mathrm{e}^{-(y_0+1)^2/(2\sigma^2)}}\right] = \ln\left[\mathrm{e}^{-[(y_0-1)^2+(y_0+1)^2]/(2\sigma^2)}\right] \;=$$
$$= \;\frac{(y_0+1)^2-(y_0-1)^2}{2\cdot\sigma^2} = \frac{2\cdot y_0}{\sigma^2}.$$

Ersetzen wir nun die Hilfsgröße y_0 durch die (allgemeine) Zufallsgröße y am AWGN–Ausgang, so lautet das Endergebnis:

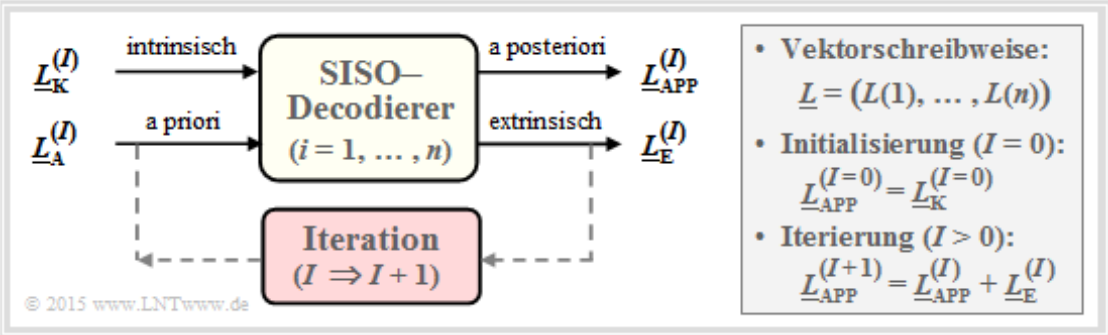
$$L(y\,|\,x) = 2\cdot y/\sigma^2 = K_{\rm L}\cdot y.$$

Hierbei ist $K_{\rm L} = 2/\sigma^2$ eine Konstante, die allein von der Streuung der Gaußschen Störung abhängt.

Symbolweise Soft–in Soft–out Decodierung

Wir gehen nun von einem (n, k)–Blockcode aus, wobei das Codewort $\underline{x} = (x_1, \ldots, x_n)$ durch den Kanal in das Empfangswort $\underline{y} = (y_1, \ldots, y_n)$ verfälscht wird. Bei langen Codes ist eine *Maximum–a–posteriori–Entscheidung auf Blockebene* – kurz: **block–wise MAP** – sehr aufwändig. Man müsste unter den 2^k zulässigen Codeworten $\underline{x}_j \in C$ dasjenige mit der größten Rückschlusswahrscheinlichkeit (englisch: *A Posteriori Probability*, APP) finden. Das auszugebende Codewort $\underline{z}$ wäre in diesem Fall

$$\underline{z} = \arg \max_{\underline{x}_j \in C} \Pr(\underline{x}_j | \underline{y}) .$$



Eine zweite Möglichkeit ist die Decodierung auf Bitebene. Der dargestellte symbolweise (oder bitweise) **Soft–in Soft–out Decoder** hat die Aufgabe, alle Codewortbits $x_i \in \{0, 1\}$ entsprechend maximaler Rückschlusswahrscheinlichkeit $\Pr(x_i \,|\, \underline{y})$ zu decodieren. Mit der Laufvariablen $i = 1, \ldots, n$ gilt dabei:

- Der entsprechende L–Wert (englisch: *Log Likelihood Ratio*, LLR) für das i–te Codebit lautet:

$$L_{\rm APP}(i) = L(x_i \,|\, \underline{y}) = \ln \frac{\Pr(x_i = 0 \,|\, \underline{y})}{\Pr(x_i = 1 \,|\, \underline{y})} .$$

- Der Decoder arbeitet iterativ. Bei der Initialisierung (gekennzeichnet durch den Parameter $I = 0$) ist $L_{\rm APP}(i) = L_{\rm K}(i)$, wobei das Kanal–LLR $L_{\rm K}(i)$ durch den Empfangswert y_i gegeben ist.
- Berechnet wird zudem der extrinsische L–Wert $L_{\rm E}(i)$, der die gesamte Information quantifiziert, die alle anderen Bits $(j \neq i)$ aufgrund der Code–Eigenschaften über das betrachtete i–te Bit liefern.
- Bei der nächsten Iteration (ab $I = 1$) wird $L_{\rm E}(i)$ bei der Berechnung von $L_{\rm APP}(i)$ als Apriori–Information $L_{\rm A}(i)$ berücksichtigt. Für das neue Aposteriori–LLR in der Iteration $I + 1$ gilt somit:

$$L_{\rm APP}^{(I+1)}(i) = L_{\rm APP}^{(I)}(i) + L_{\rm A}^{(I+1)}(i) = L_{\rm APP}^{(I)}(i) + L_{\rm E}^{(I)}(i) .$$

- Die Iterationen werden fortgesetzt, bis alle $|L_{\rm APP}(i)|$ größer sind als ein vorzugebender Wert. Das wahrscheinlichste Codewort $\underline{z}$ ergibt sich dann aus den Vorzeichen aller $L_{\rm APP}(i)$, mit $i = 1, \ldots, n$.
- Bei einem **systematischen Code** geben die ersten k Bit von $\underline{z}$ das gesuchte Informationswort an, das mit großer Wahrscheinlichkeit mit der gesendeten Nachricht $\underline{u}$ übereinstimmen wird.

Diese Beschreibung des SISO–Decodierers nach **[Bos98]** soll in erster Linie die unterschiedlichen L–Werte verdeutlichen. Das große Potential der symbolweisen Decodierung erkennt man allerdings erst im Zusammenhang mit **verketteten Codiersystemen.**

Zur Berechnung der extrinsischen L–Werte (1)

Die Schwierigkeit bei der symbolweisen iterativen Decodierung ist im allgemeinen die Bereitstellung der extrinsischen L–Werte $L_{\rm E}(i)$. Bei einem Code der Länge n gilt hierbei für die Laufvariable: $i = 1, \ ... \ , n$.

Definition: Der **extrinsische L–Wert** (englisch: *extrinsic LLR*) ist ein Maß für die Informationen, den die anderen Symbole ($j \neq i$) des Codewortes über das i–te Codesymbol liefern, ausgedrückt als Log–Likelihood–Verhältnis. Wir benennen diesen Kennwert mit $L_{\rm E}(i)$.

Wir berechnen nun die extrinsischen L–Werte $L_{\rm E}(i)$ für zwei beispielhafte Codes.

Repetition Code ⇒ RC (n, 1, n)

Ein Wiederholungscode zeichnet sich dadurch aus, dass alle n Codesymbole $x_i \in \{0, 1\}$ identisch sind. Der extrinsische L–Wert für das i–ten Symbol ist hier sehr einfach anzugeben und lautet:

$$L_{\rm E}(i) = \sum_{j \neq i} L_j \quad \text{mit} \quad L_j = L_{\rm APP}(j).$$

Ist die Summe über alle $L_{j \neq i}$ positiv, so bedeutet dies aus Sicht der anderen L–Werte eine Präferenz für „$x_i = 0$". Bei negativer Summe ist „$x_i = 1$" wahrscheinlicher. $L_{\rm E}(i) = 0$ erlaubt keine Vorhersage.

Beispiel: Wir betrachten die Decodierung eines Wiederholungscodes ($n = 4$), wobei gelten soll:

- $\underline{L} = (+1, -1, +3, -1) \Rightarrow \underline{L}_{\rm E} = (+1, +3, -1, +3)$: $L_{\rm E}(1)$ ist positiv, obwohl zwei der anderen L–Werte (Mehrheit) negativ sind ⇒ Präferenz für „$x_1 = 0$". Alle Aposteriori–L–Werte sind nach einer Iteration positiv ⇒ Informationsbit ist $u = 0$. Weitere Iterationen bringen nichts.

	$L_{\rm APP}(1)$	$L_{\rm APP}(2)$	$L_{\rm APP}(3)$	$L_{\rm APP}(4)$	$L_{\rm E}(1)$	$L_{\rm E}(2)$	$L_{\rm E}(3)$	$L_{\rm E}(4)$	Bemerkungen
$I = 0$	+1	–1	+3	–1	+1	+3	–1	+3	©2015 www.LNTwww.de
$I = 1$	+2	+2	+2	+2					alle Vorzeichen von $L_{\rm APP}(i)$ sind erstmals richtig

- $\underline{L} = (+1, +1, -4, +1) \Rightarrow \underline{L}_{\rm E} = (-2, -2, +3, -2)$: Alle Aposteriori–$L$–Werte sind nach zwei Iterationen negativ ⇒ Informationsbit ist $u = 1$ ⇒ zu Beginn waren drei Vorzeichen falsch.

	$L_{\rm APP}(1)$	$L_{\rm APP}(2)$	$L_{\rm APP}(3)$	$L_{\rm APP}(4)$	$L_{\rm E}(1)$	$L_{\rm E}(2)$	$L_{\rm E}(3)$	$L_{\rm E}(4)$	Bemerkungen
$I = 0$	+1	+1	–4	+1	–2	–2	+3	–2	©2015 www.LNTwww.de
$I = 1$	–1	–1	+1	–1	–1	–1	–3	–1	
$I = 2$	–2	–2	–2	–2					alle Vorzeichen von $L_{\rm APP}(i)$ sind erstmals richtig

- $\underline{L} = (+1, +1, -3, +1) \Rightarrow \underline{L}_{\rm E} = (-1, -1, +3, -1)$: Alle Aposteriori–$L$–Werte sind nach einer Iteration 0 ⇒ Informationsbit kann nicht decodiert werden. Weitere Iterationen bringen nichts.

	$L_{\rm APP}(1)$	$L_{\rm APP}(2)$	$L_{\rm APP}(3)$	$L_{\rm APP}(4)$	$L_{\rm E}(1)$	$L_{\rm E}(2)$	$L_{\rm E}(3)$	$L_{\rm E}(4)$	Bemerkungen
$I = 0$	+1	+1	–3	+1	–1	–1	+3	–1	©2015 www.LNTwww.de
$I = 1$	0	0	0	0					Decodierung versagt

Zur Berechnung der extrinsischen L–Werte (2)

Single Parity–check Code ⇒ SPC (n, n –1, 2)

Bei jedem *Single Parity–check Code* ist die Anzahl der Einsen in jedem Codewort geradzahlig. Oder anders ausgedrückt: Für jedes Codewort $\underline{x}$ ist das **Hamming–Gewicht** $w_{\rm H}(\underline{x})$ geradzahlig.

Das Codewort $\underline{x}^{(-i)}$ beinhalte alle Symbole mit Ausnahme von x_i ⇒ Vektor der Länge $n-1$. Damit lautet der extrinsische L–Wert bezüglich dieses i–ten Symbols, wenn $\underline{x}$ empfangen wurde:

$$L_{\rm E}(i) = \ln \frac{\Pr\left[w_{\rm H}(\underline{x}^{(-i)}) \text{ ist gerade} \,|\, \underline{y}\right]}{\Pr\left[w_{\rm H}(\underline{x}^{(-i)}) \text{ ist ungerade} \,|\, \underline{y}\right]}.$$

Wie in der **Aufgabe A4.4** gezeigt werden soll, kann hierfür auch geschrieben werden:

$$L_{\rm E}(i) = 2 \cdot \tanh^{-1}\left[\prod_{j \neq i}^{n} \tanh(L_j/2)\right] \quad \text{mit} \quad L_j = L_{\rm APP}(j).$$

Beispiel: Wir gehen vom *Single Parity–check Code* mit $n = 3$, $k = 2$ ⇒ kurz **SPC (3, 2, 2)** aus. Die $2^k = 4$ gültigen Codeworte $\underline{x} = \{x_1, x_2, x_3\}$ lauten bei bipolarer Beschreibung ⇒ $x_i \in \{\pm 1\}$:

$$\underline{x}_0 = (+1,+1,+1), \quad \underline{x}_1 = (+1,-1,-1), \quad \underline{x}_2 = (-1,+1,-1), \quad \underline{x}_3 = (-1,-1,+1).$$

Bei diesem Code ist also das Produkt $x_1 \cdot x_2 \cdot x_3$ stets positiv.

	$L_{\rm APP}(1)$	$L_{\rm APP}(2)$	$L_{\rm APP}(3)$	$L_{\rm E}(1)$	$L_{\rm E}(2)$	$L_{\rm E}(3)$	Bemerkungen
$I = 0$	+2.000	+0.400	–1.600	–0.131	–0.518	+0.151	© 2016 www.LNTwww.de
$I = 1$	+1.869	–0.118	–1.449				alle Vorzeichen von $L_{\rm APP}(i)$ sind erstmals richtig

Die Tabelle zeigt den Decodiervorgang für $\underline{L}_{\rm APP}$ = (+2.0, +0.4, –1.6). Die harte Entscheidung nach den Vorzeichen von $L_{\rm APP}(i)$ ergäbe hier (+1, +1, –1), also kein gültiges Codewort des SP(3, 2, 2).

Rechts sind in der Tabelle die dazugehörigen extrinsischen L–Werte eingetragen:

$$L_{\rm E}(1) = 2 \cdot \tanh^{-1}\left[\tanh(0.2) \cdot \tanh(-0.8)\right] = -0.131,$$
$$L_{\rm E}(2) = 2 \cdot \tanh^{-1}\left[\tanh(1.0) \cdot \tanh(-0.8)\right] = -0.518,$$
$$L_{\rm E}(3) = 2 \cdot \tanh^{-1}\left[\tanh(1.0) \cdot \tanh(0.2)\right] = +0.151.$$

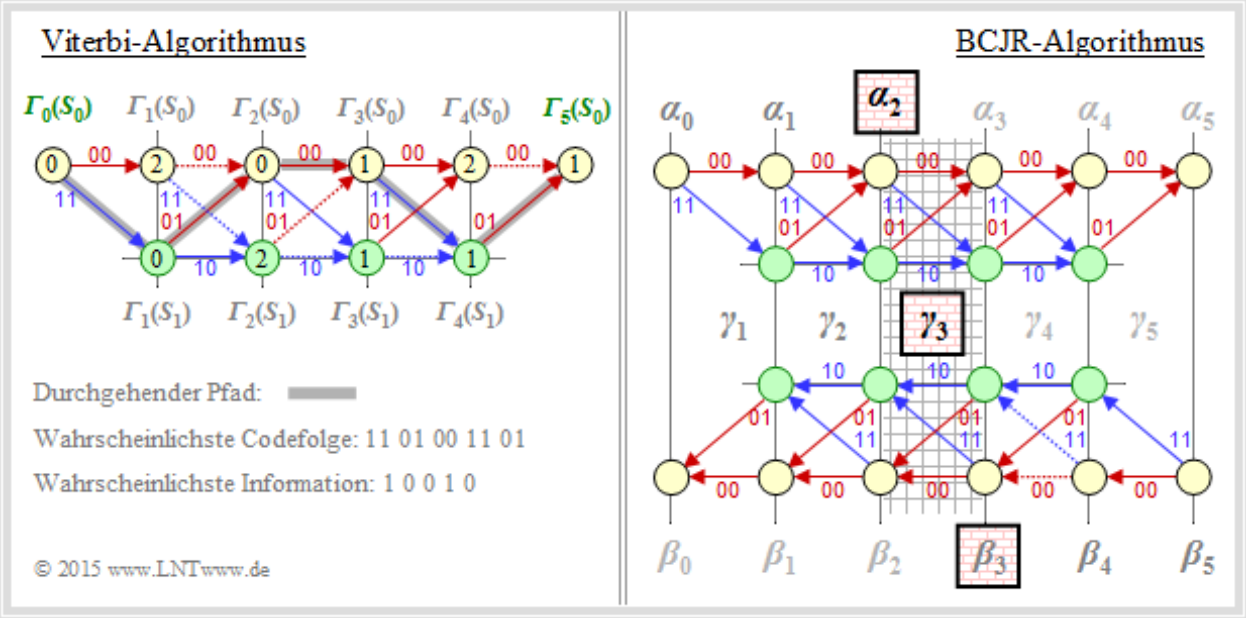
Die zweite Gleichung lässt sich wie folgt interpretieren:

- $L_{\rm APP}(1)$ = +2.0 und $L_{\rm APP}(3)$ = –1.6 sagen aus, dass Bit 1 eher „+1" als „–1" ist und Bit 3 eher „–1" als „+1". Die Zuverlässigkeit (Betrag) ist beim ersten Bit etwas größer als beim dritten.
- Die extrinsische Information $L_{\rm E}(2)$ = –0.518 berücksichtigt nur die Informationen von Bit 1 und Bit 3 über Bit 2. Aus deren Sicht ist das zweite Bit eine „–1" mit der Zuverlässigkeit 0.518.
- Der vom Empfangswert y_2 abgeleitete L–Wert ⇒ $L_{\rm APP}(2)$ = +0.4 hat für das zweite Bit eine „+1" vermuten lassen. Die Diskrepanz wird hier bereits in der Iteration $I = 1$ aufgelöst.
- Entschieden wird hier für das Codewort $\underline{x}_1$. Bei $0.518 < L_{\rm APP}(2) < 1.6$ würde das Ergebnis $\underline{x}_1$ erst nach mehreren Iterationen vorliegen. Für $L_{\rm APP}(2) > 1.6$ liefert der Decoder dagegen $\underline{x}_0$.

BCJR–Decodierung: Vorwärts–Rückwärts–Algorithmus

Ein Beispiel für die iterative Decodierung von Faltungscodes ist der *BCJR–Algorithmus*, benannt nach dessen Erfindern L. R. Bahl, J. Cocke, F. Jelinek und J. Raviv ⇒ **[BCJR74].** Der Algorithmus weist viele Parallelen zur sieben Jahren älteren Viterbi–Decodierung auf, doch auch signifikante Unterschiede:

- Während Viterbi die Gesamtsequenz schätzt ⇒ **block–wise Maximum Likelihood,** minimiert der BCJR–Algorithmus die Bitfehlerwahrscheinlichkeit ⇒ **bit–wise MAP.**
- Der Viterbi–Algorithmus kann (in seiner ursprünglichen Form) keine Softinformation verarbeiten. Dagegen gibt der BCJR–Algorithmus bei jeder Iteration für jedes einzelne Symbol (Bit) einen Zuverlässigkeitswert an, der bei späteren Iterationen berücksichtigt wird.



Die Abbildung soll – fast unzulässig vereinfacht – die unterschiedliche Vorgehensweise von Viterbi–Algorithmus (links) und BCJR–Algorithmus (rechts) verdeutlichen. Zugrunde liegt ein Faltungscode mit dem Gedächtnis $m = 1$ und der Länge $L = 4$ ⇒ Gesamtlänge $L' = 5$ (mit Terminierung).

- Der Viterbi–Algorithmus sucht und findet den wahrscheinlichsten Pfad von $\Gamma_0(S_0)$ nach $\Gamma_5(S_0)$, nämlich $S_0 \rightarrow S_1 \rightarrow S_0 \rightarrow S_0 \rightarrow S_1$. Wir verweisen auf die **Musterlösung zur Aufgabe Z3.9.**

Die Skizze für den BCJR–Algorithmus verdeutlicht die Gewinnung des extrinsischen L–Wertes für das dritte Symbol ⇒ $L_E(3)$. Der relevante Bereich im Trellis ist schraffiert:

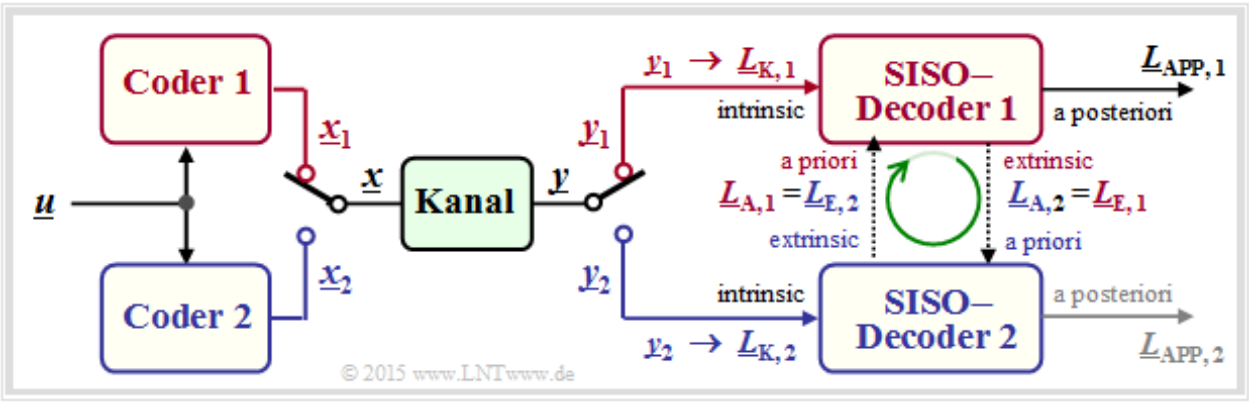
- Bei der Abarbeitung des Trellisdiagramms in Vorwärtsrichtung gewinnt man – in gleicher Weise wie bei Viterbi – die Metriken α_1, α_2, ... α_5. Zur Berechnung von $L_E(3)$ benötigt man hiervon α_2.
- Anschließend durchläuft man das Trellisdiagramm rückwärts (also von rechts nach links) und erhält damit die Metriken β_4, β_3, ... , β_0 entsprechend der unteren Skizze.
- Der gesuchte extrinsische L–Wert $L_E(3)$ ergibt sich aus den Metriken α_2 (in Vorwärtsrichtung) und β_3 (in Rückwärtsrichtung) sowie der Apriori–Information γ_3 über das Symbol $i = 3$.

Grundstruktur von verketteten Codiersystemen

Die wichtigsten Kommunikationssysteme der letzten Jahre verwenden zwei unterschiedliche Kanalcodes. Man spricht dann von **verketteten Codiersystemen** (englisch: *Concatenated Codes*). Auch bei relativ kurzen Komponentencodes C_1 und C_2 ergibt sich für den verketteten Code C eine hinreichend große Codewortlänge n, die ja bekanntlich erforderlich ist, um sich der Kanalkapazität anzunähern.

Zunächst seien einige Beispiele aus dem Mobilfunk genannt:

- Bei **GSM** (*Global System for Mobile Communications*, zweite Mobilfunkgeneration) wird zunächst die Datenbitrate von 9.6 kbit/s auf 12 kbit/s erhöht, um auch in leitungsvermittelten Netzen eine Fehlererkennung zu ermöglichen. Anschließend folgt ein punktierter Faltungscode mit der Ausgangsbitrate 22.8 kbit/s. Die Gesamtcoderate beträgt somit etwa 42.1%.
- Beim 3G–Mobilfunksystem **UMTS** (*Universal Mobile Telecommunications System*) verwendet man je nach den Randbedingungen (guter/schlechter Kanal, wenige/viele Teilnehmer in der Zelle) einen **Faltungscode** oder einen **Turbocode** (darunter versteht man per se die Verkettung zweier Faltungscodierer). Beim 4G–Mobilfunksystem **LTE** (*Long Term Evolution*) verwendet man für kurze Kontrollsignale einen Faltungscode und für die längeren Payload-Daten einen Turbocode.



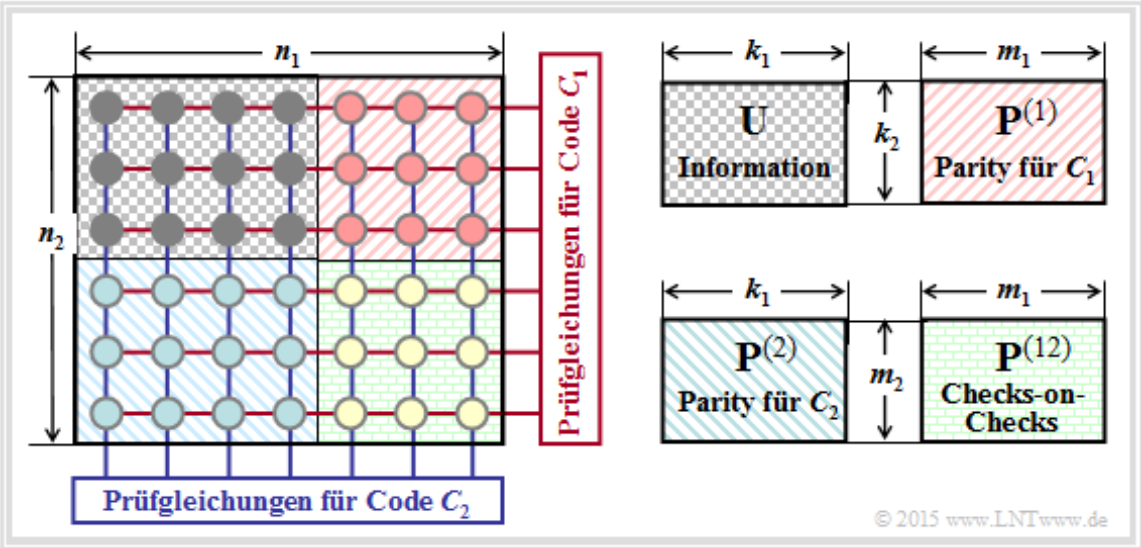
Die Grafik zeigt die Grundstruktur eines parallel verketteten Codiersystems. Alle Vektoren bestehen aus n Elementen: $\underline{L} = (L(1), \ \ldots \ , L(n))$. Die Berechnung aller L–Werte geschieht also auf Symbolebene. Nicht dargestellt ist hier der **Interleaver,** der zum Beispiel bei den Turbocodes obligatorisch ist.

- Die Codesequenzen $\underline{x}_1$ und $\underline{x}_2$ werden zur gemeinsamen Übertragung über den Kanal durch einen Multiplexer zum Vektor $\underline{x}$ zusammengefügt. Am Empfänger wird die Sequenz $\underline{y}$ wieder in die Einzelteile $\underline{y}_1$ und $\underline{y}_2$ zerlegt. Daraus werden die Kanal–L–Werte $\underline{L}_{\rm K,1}$ und $\underline{L}_{\rm K,2}$ gebildet.
- Der symbolweise Decoder ermittelt entsprechend der vorne beschriebenen **Vorgehensweise** die extrinsischen L–Werte $\underline{L}_{\rm E,1}$ und $\underline{L}_{\rm E,2}$, die gleichzeitig die Apriori–Informationen $\underline{L}_{\rm A,2}$ und $\underline{L}_{\rm A,1}$ für den jeweils anderen Decoder darstellen.
- Nach ausreichend vielen Iterationen (also dann, wenn ein Abbruchkriterium erfüllt ist) liegt am Decoderausgang der Vektor der Aposteriori–Werte $\Rightarrow$ $\underline{L}_{\rm APP}$ an. Im Beispiel wird willkürlich der Wert im oberen Zweig genommen. Möglich wäre aber auch der untere L–Wert.

Das obige Modell gilt insbesondere auch für die Decodierung der Turbo–Codes gemäß **Kapitel 4.3.**

Grundstruktur eines Produktcodes

Die Grafik zeigt den prinzipiellen Aufbau von Produktcodes, die bereits 1954 von **Peter Elias** eingeführt wurden. Der nachfolgend dargestellte **zweidimensionale Produktcode** $C = C_1 \times C_2$ basiert auf den beiden linearen und binären Blockcodes mit den Parametern (n_1, k_1) bzw. (n_2, k_2).



Die Codewortlänge ist $n = n_1 \cdot n_2$. Diese n Codebits lassen sich wie folgt gruppieren:

- Die $k = k_1 \cdot k_2$ Informationsbits sind in der $k_2 \times k_1$–Matrix $\mathbf{U}$ angeordnet. Die Coderate ist gleich dem Produkt der Coderaten der beiden Basiscodes: $R = k/n = (k_1/n_1) \cdot (k_2/n_2) = R_1 \cdot R_2$.
- Die rechte obere Matrix $\mathbf{P}^{(1)}$ mit der Dimension $k_2 \times m_1$ beinhaltet die Prüfbits (englisch: *Parity*) hinsichtlich des Codes C_1. In jeder der k_2 Zeilen werden zu den k_1 Informationsbits $m_1 = n_1 - k_1$ Prüfbits hinzugefügt, wie in **Kapitel 1.3** am Beispiel der Hamming–Codes beschrieben wurde.
- Die linke untere Matrix $\mathbf{P}^{(2)}$ der Dimension $m_2 \times k_1$ beinhaltet die Prüfbits hinsichtlich des zweiten Komponentencodes C_2. Hier erfolgt die Codierung (und auch die Decodierung) zeilenweise: In jeder der k_1 Spalten werden die k_2 Informationsbits noch um $m_2 = n_2 - k_2$ Prüfbits ergänzt.
- Die $m_2 \times m_1$–Matrix $\mathbf{P}^{(12)}$ rechts unten bezeichnet man als *Checks–on–Checks*. Hier werden die vorher erzeugten Parity–Matrizen $\mathbf{P}^{(1)}$ und $\mathbf{P}^{(2)}$ entsprechend den Prüfgleichungen verknüpft.

Alle Produktcodes entsprechend obiger Grafik weisen folgende **Eigenschaften** auf:

- Bei linearen Komponentencodes C_1 und C_2 ist auch der Produktcode $C = C_1 \times C_2$ linear.
- Jede Zeile von C gibt ein Codewort von C_1 wieder und jede Spalte ein Codewort von C_2.
- Die Summe zweier Zeilen ergibt aufgrund der Linearität wieder ein Codewort von C_1.
- Ebenso ergibt die Summe zweier Spalten ein gültiges Codewort von C_2.
- Jeder Produktcodes beinhaltet auch das Nullwort $\underline{0}$ (ein Vektor mit n Nullen).
- Die minimale Distanz von C ist $d_{\rm min} = d_1 \cdot d_2$, wobei d_i die minimale Distanz von C_i angibt.

Iterative Syndromdecodierung von Produktcodes (1)

Wir betrachten nun den Fall, dass ein Produktcode mit Matrix **X** über einen Binärkanal übertragen wird. Die Empfangsmatrix sei **Y** = **X** + **E**, wobei **E** die Fehlermatrix bezeichnet. Alle Elemente der Matrizen **X**, **E** und **Y** seien binär, also 0 oder 1.

Für die Decodierung der beiden Komponentencodes bietet sich die Syndromdecodierung entsprechend dem **Kapitel 1.5** an. Im zweidimensionalen Fall bedeutet dies:

- Man decodiert zunächst die n_2 Zeilen der Empfangsmatrix **Y**, basierend auf der Prüfmatrix $\mathbf{H}_1$ des Komponentencodes C_1. Man verwendet hierzu die Syndromdecodierung.
- Dazu bildet man jeweils das sogenannte Syndrom $\underline{s} = \underline{y} \cdot \mathbf{H}_1^{\rm T}$, wobei der Vektor $\underline{y}$ der Länge n_1 die aktuelle Zeile von **Y** angibt und „T" für „transponiert" steht.
- Entsprechend dem berechneten $\underline{s}_\mu$ (mit $0 \le \mu < 2^{n_1-k_1}$) findet man dann in einer vorbereiteten Syndromtabelle das wahrscheinliche Fehlermuster $\underline{e} = \underline{e}_\mu$.
- Bei nur wenigen Fehlern innerhalb der Zeile stimmt dann $\underline{y} + \underline{e}$ mit dem gesendeten Zeilenvektor $\underline{x}$ überein. Sind zu viele Fehler aufgetreten, so kommt es allerdings zu Fehlkorrekturen.
- Anschließend syndromdecodiert man die n_1 Spalten der (korrigierten) Empfangsmatrix **Y'**, diesmal basierend auf der (transponierten) Prüfmatrix $\mathbf{H}_2^{\rm T}$ des Komponentencodes C_2.
- Hierzu bildet man das Syndrom $\underline{s} = \underline{y}' \cdot \mathbf{H}_2^{\rm T}$, wobei der Vektor $\underline{y}'$ der Länge n_2 die betrachtete Spalte von **Y'** bezeichnet.
- Aus einer zweiten Syndromtabelle (gültig für den Code C_2) findet man für das berechnete $\underline{s}_\mu$ (mit $0 \le \mu < 2^{n_2-k_2}$) das wahrscheinliche Fehlermuster $\underline{e} = \underline{e}_\mu$ der bearbeiteten Spalte.
- Nach Korrektur aller Spalten liegt die Marix **Y''** vor. Nun kann man wieder eine Zeilen– und anschließend eine Spaltendecodierung vornehmen ⇒ zweite Iteration, und so weiter, und so fort.

Auf der nächsten Seite wird dieser Decodieralgorithmus an einem Beispiel verdeutlicht. Dazu betrachten wir wieder den (42, 12) Produktcode, basierend auf

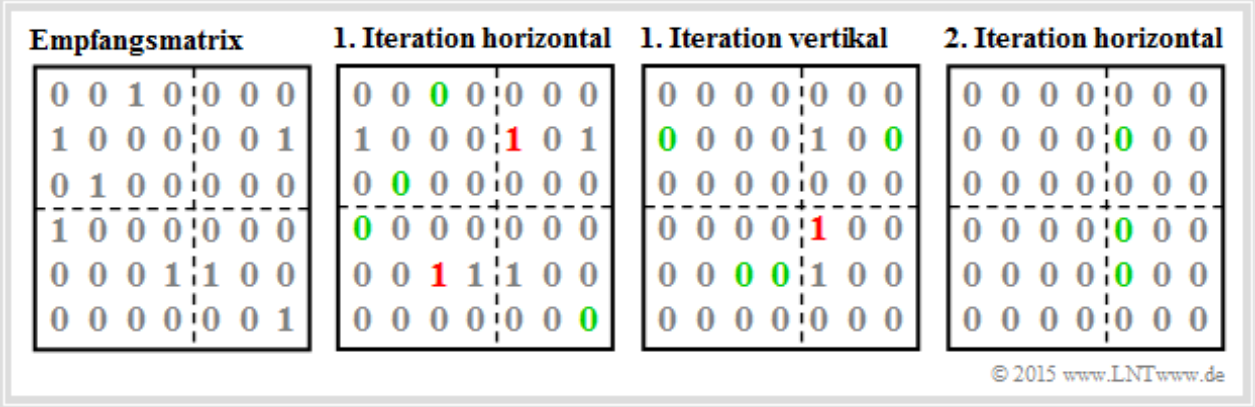
- dem Hammingcode (7, 4, 3) ⇒ Code C_1,
- dem verkürzten Hammingcode (6, 3, 3) ⇒ Code C_2.

Die Prüfmatrizen dieser beiden systematischen Komponentencodes lauten in transponierter Form:

$$\mathbf{H}_1^{\rm T} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{H}_2^{\rm T} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Iterative Syndromdecodierung von Produktcodes (2)

Die linke Grafik zeigt die Empfangsmatrix **Y**, Aus Darstellungsgründen wurde die Codermatrix **X** zu einer 6 × 7–Nullmatrix gewählt, so dass die neun Einsen in **Y** gleichzeitig Übertragungsfehler darstellen.



Hinweis: Für die folgenden Berechnungen ist der Link zu den **transponierten Prüfmatrizen** nützlich.

Die **zeilenweise Syndromdecodierung** geschieht über das Syndrom $\underline{s} = \underline{y} \cdot \mathbf{H}_1^{\rm T}$. Im Einzelnen:

- **Zeile 1** ⇒ Einzelfehlerkorrektur ist erfolgreich (ebenso in den Zeilen 3, 4 und 6):

$$\underline{s} = (0, 0, 1, 0, 0, 0, 0) \cdot \mathbf{H}_1^{\rm T} = (0, 1, 1) = \underline{s}_3$$
$$\Rightarrow \quad \underline{y} + \underline{e}_3 = (0, 0, 0, 0, 0, 0, 0) \; .$$

- **Zeile 2** ⇒ Fehlkorrektur bezüglich Bit 5:

$$\underline{s} = (1, 0, 0, 0, 0, 0, 1) \cdot \mathbf{H}_1^{\rm T} = (1, 0, 0) = \underline{s}_4$$
$$\Rightarrow \quad \underline{y} + \underline{e}_4 = (1, 0, 0, 0, 1, 0, 1) \; .$$

- **Zeile 5** ⇒ Fehlkorrektur bezüglich Bit 3:

$$\underline{s} = (0, 0, 0, 1, 1, 0, 0) \cdot \mathbf{H}_1^{\rm T} = (0, 1, 1) = \underline{s}_3$$
$$\Rightarrow \quad \underline{y} + \underline{e}_3 = (0, 0, 1, 1, 1, 0, 0) \; .$$

Hammingcode (7, 4, 3)

Syndrom $\underline{s}_\mu$	Nebenklassenanführer $\underline{e}_\mu$
$\underline{s}_0 = (0, 0, 0)$	$\underline{e}_0 = (0, 0, 0, 0, 0, 0, 0)$
$\underline{s}_1 = (0, 0, 1)$	$\underline{e}_1 = (0, 0, 0, 0, 0, 0, 1)$
$\underline{s}_2 = (0, 1, 0)$	$\underline{e}_2 = (0, 0, 0, 0, 0, 1, 0)$
$\underline{s}_3 = (0, 1, 1)$	$\underline{e}_3 = (0, 0, 1, 0, 0, 0, 0)$
$\underline{s}_4 = (1, 0, 0)$	$\underline{e}_4 = (0, 0, 0, 0, 1, 0, 0)$
$\underline{s}_5 = (1, 0, 1)$	$\underline{e}_5 = (1, 0, 0, 0, 0, 0, 0)$
$\underline{s}_6 = (1, 1, 0)$	$\underline{e}_6 = (0, 1, 0, 0, 0, 0, 0)$
$\underline{s}_7 = (1, 1, 1)$	$\underline{e}_7 = (0, 0, 0, 1, 0, 0, 0)$

Die **spaltenweisen Syndromdecodierung** entfernt alle Einzelfehler in den Spalten 1, 2, 3, 4, 6 und 7.

- **Spalte 5** ⇒ Fehlkorrektur bezüglich Bit 4:

$$\underline{s} = (0, 1, 0, 0, 1, 0) \cdot \mathbf{H}_1^{\rm T} = (1, 0, 0) = \underline{s}_4$$
$$\Rightarrow \quad \underline{y} + \underline{e}_4 = (0, 1, 0, 1, 1, 0) \; .$$

Die verbliebenen drei Fehler werden durch zeilenweise Decodierung der **zweiten Iterationsschleife** korrigiert.

Ob alle Fehler eines Blockes korrigierbar sind, hängt vom Fehlermuster ab. Hier verweisen wir auf **Aufgabe A4.7.**

Verkürzter Hammingcode (6, 3, 3)

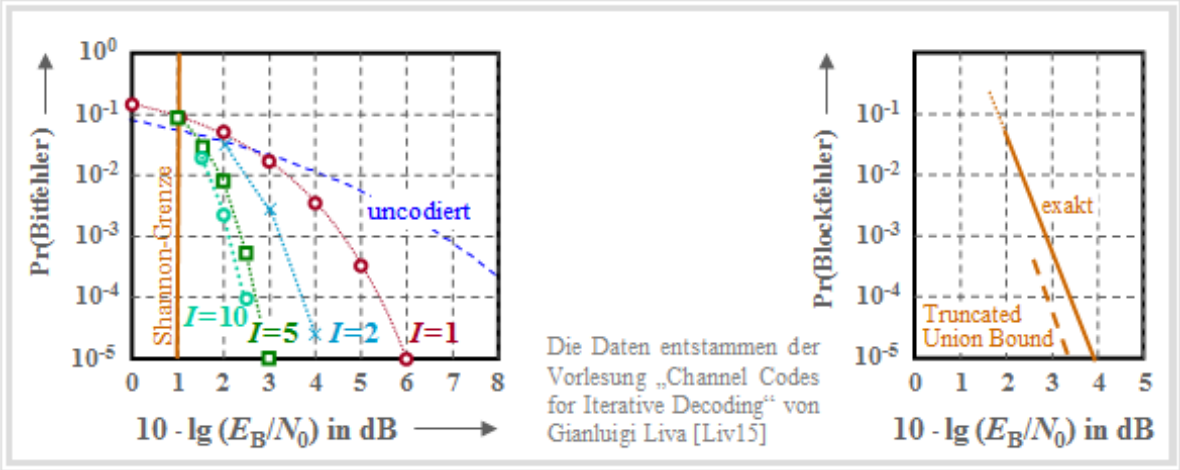
Syndrom $\underline{s}_\mu$	Nebenklassenanführer $\underline{e}_\mu$
$\underline{s}_0 = (0, 0, 0)$	$\underline{e}_0 = (0, 0, 0, 0, 0, 0)$
$\underline{s}_1 = (0, 0, 1)$	$\underline{e}_1 = (0, 0, 0, 0, 0, 1)$
$\underline{s}_2 = (0, 1, 0)$	$\underline{e}_2 = (0, 0, 0, 0, 1, 0)$
$\underline{s}_3 = (0, 1, 1)$	$\underline{e}_3 = (0, 0, 1, 0, 0, 0)$
$\underline{s}_4 = (1, 0, 0)$	$\underline{e}_4 = (0, 0, 0, 1, 0, 0)$
$\underline{s}_5 = (1, 0, 1)$	$\underline{e}_5 = (0, 1, 0, 0, 0, 0)$
$\underline{s}_6 = (1, 1, 0)$	$\underline{e}_6 = (1, 0, 0, 0, 0, 0)$
$\underline{s}_7 = (1, 1, 1)$	$\underline{e}_7 = (1, 0, 0, 0, 0, 1)$

Leistungsfähigkeit der Produktcodes

Die 1954 eingeführten **Produktcodes** waren die ersten Codes, die auf rekursiven Konstruktionsregeln basierten und somit grundsätzlich für die iterative Decodierung geeignet waren. Der Erfinder **Peter Elias** hat sich diesbezüglich zwar nicht geäußert, aber in den letzten zwanzig Jahren hat dieser Aspekt und die gleichzeitige Verfügbarkeit schneller Prozessoren dazu beigetragen, dass inzwischen auch Produktcodes in realen Kommunikationssystemen eingesetzt werden, z. B. beim Fehlerschutz von Speichermedien und bei Glasfasersystemen mit sehr hoher Datenrate.

Meist verwendet man sehr lange Produktcodes (großes $n = n_1 \cdot n_2$) mit folgender Konsequenz:

- Aus Aufwandsgründen ist hier die **Maximum–Likelihood–Decodierung auf Blockebene** für die Komponentencodes C_1 und C_2 nicht anwendbar, auch nicht die **Syndromdecodierung,** die ja eine Realisierungsform der ML–Decodierung darstellt.
- Anwendbar ist dagegen auch bei großem n die iterative symbolweise MAP–Decodierung, wie in **Kapitel 4.1** beschrieben. Der Austausch von extrinsischer und Apriori–Information geschieht hier zwischen den beiden Komponentencodes. Genaueres hierüber findet man in **[Liv15].**



Die Grafik zeigt für einen (1024, 676)–Produktcode, basierend auf dem *Extended Hammingcode* eHC (32, 26) als Komponentencodes, die AWGN–Bitfehlerwahrscheinlichkeit (links) in Abhängigkeit der Iterationen (I) sowie rechts die Fehlerwahrscheinlichkeit der Blöcke (bzw. Codeworte). Die Coderate beträgt $R = R_1 \cdot R_2 = 0.66$, womit sich die Shannon–Grenze zu $10 \cdot \lg (E_B/N_0) \approx 1$ dB ergibt.

Links erkennt man den Einfluss der Iterationen. Beim Übergang von $I = 1$ auf I gewinnt man ca. 2 dB (bei der Bitfehlerrate 10^{-5}) und mit $I = 10$ ein weiteres dB. Weitere Iterationen lohnen sich nicht.

Auch alle in **Kapitel 1.6** genannten Schranken können hier angewendet werden, so auch die in der rechten Grafik gestrichelt eingezeichneten *Truncated Union Bound*:

$$\Pr(\text{Truncated0.15cm Union Bound}) = W_{d_{\min}} \cdot \text{Q}\left(\sqrt{d_{\min} \cdot 2R \cdot E_B/N_0}\right) .$$

Die minimale Distanz beträgt $d_{\min} = d_1 \cdot d_2 = 4 \cdot 4 = 16$. Mit der Gewichtsfunktion des eHC (32, 26),

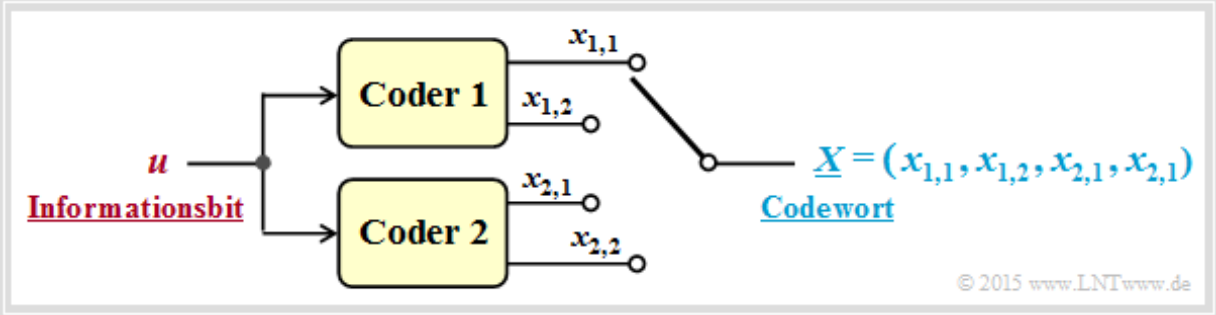
$$W_{\text{eHC}(32,\,26)}(X) = 1 + 1240 \cdot X^4 + 27776 \cdot X^6 + 330460 \cdot X^8 + \ldots + X^{32},$$

erhält man für den Produktcode $W_{d,\,\min} = 1240^2 = 15376000$. Damit ist die obere Gleichung bestimmt.

Grundstruktur eines Turbocodes (1)

Alle heute (2016) aktuellen Kommunikationssysteme wie **UMTS** (3. Mobilfunkgeneration) und **LTE** (4. Mobilfunkgeneration) verwenden das in **Kapitel 4.1** dargelegte Konzept der symbolweisen iterativen Decodierung. Dass dies so ist, steht unmittelbar mit der Erfindung der Turbocodes im Jahre 1993 durch C. Berrou, A. Glavieux und P. Thitimajshima in Zusammenhang, denn erst mit diesen Codes konnte man sich der Shannon–Grenze mit vertretbarem Decodieraufwand annähern.

Turbocodes ergeben sich durch die parallele oder serielle Verkettung von Faltungscodes. Die Grafik zeigt die parallele Verkettung zweier Codes mit den Parametern $k = 1$, $n = 2$ $\Rightarrow$ Rate $R = 1/2$.

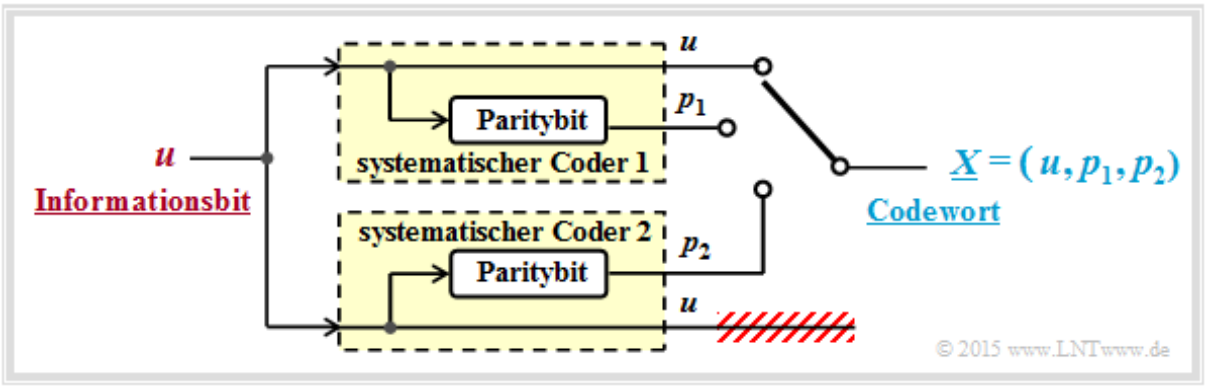


In dieser Darstellung bezeichnet:

- u das aktuell betrachtete Bit der Informationssequenz $\underline{u}$,
- $x_{i,j}$ das aktuell betrachtete Bit am Ausgang j von Coder i (mit $1 \le i \le 2$, $1 \le j \le 2$),
- $\underline{X} = (x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2})$ das Codewort für das aktuelle Informationsbit u.

Die resultierende Rate des verketteten Codiersystems ist somit $R = 1/4$.

Verwendet man systematische Komponentencodes, so ergibt sich das folgende Modell:



Die Modifikationen gegenüber der oberen Grafik lassen sich wie folgt begründen:

- Bei systematischen Codes C_1 und C_2 ist sowohl $x_{1,1} = u$ als auch $x_{2,1} = u$. Deshalb kann man auf die Übertragung eines redundanten Bits (zum Beispiel $x_{2,2}$) verzichten.
- Mit dieser Reduktion ergibt sich ein Rate–1/3–Turbocode mit den Parametern $k = 1$ und $n = 3$. Das Codewort lautet mit den Paritybits p_1 und p_2 von Coder 1 bzw. Coder 2:

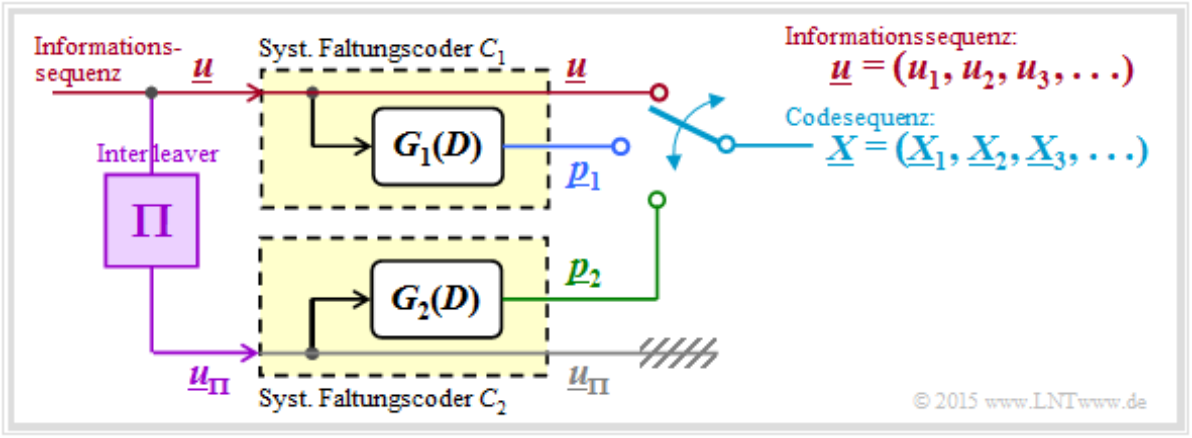
$$\underline{X} = (u, p_1, p_2) \ .$$

Auf der nächsten Seite wird unser Turbocode–Modell nochmals geringfügig modifiziert.

Grundstruktur eines Turbocodes (2)

Im Folgenden gehen wir stets von einem noch weiter modifizierten Turbocoder–Modell aus:

- Wie es für die Beschreibung von Faltungscodes erforderlich ist, liegt nun am Eingang anstelle des isolierten Informationsbits u die Informationssequenz $\underline{u} = (u_1, u_2, \text{ ... }, u_i, \text{ ... })$ an.
- Die Codewortsequenz wird mit $\underline{x} = (\underline{X}_1, \underline{X}_2, \text{ ... }, \underline{X}_i, \text{ ... })$ bezeichnet. Um Verwechslungen zu vermeiden, wurden vorne die Codeworte $\underline{X}_i = (u, p_1, p_2)_i$ mit Großbuchstaben eingeführt.

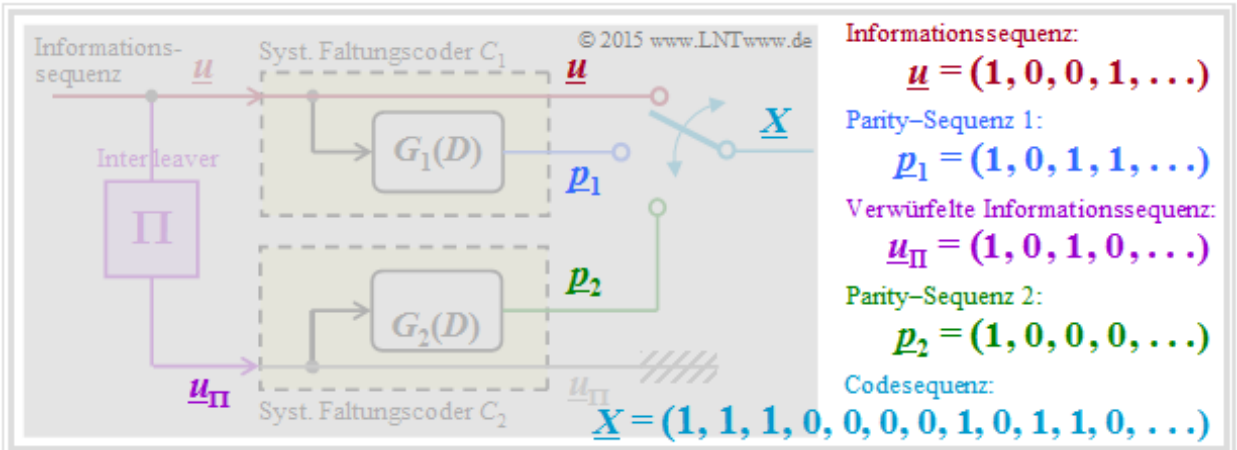


- Die Coder C_1 und C_1 werden (gedanklich) als **Digitale Filter** konzipiert und sind somit durch die **Übertragungsfunktionen** $G_1(D)$ und $G_2(D)$ darstellbar.
- Aus verschiedenen Gründen ⇒ siehe **übernächste Seite** sollte die Eingangssequenz des zweiten Coders ⇒ $\underline{u}_\pi$ gegenüber der Sequenz $\underline{u}$ durch einen Interleaver (Π) verwürfelt sein.
- Somit spricht nichts dagegen, die beiden Coder gleich zu wählen: $G_1(D) = G_2(D) = G(D)$. Ohne Interleaver würde dies die Korrekturfähigkeit extrem einschränken.

Beispiel: Die Grafik zeigt die verschiedenen Sequenzen in angepassten Farben. Anzumerken ist:

(1) Für $\underline{u}_\pi$ ist eine 3×4–Interleaver–Matrix entsprechend **Aufgabe Z4.8** berücksichtigt.

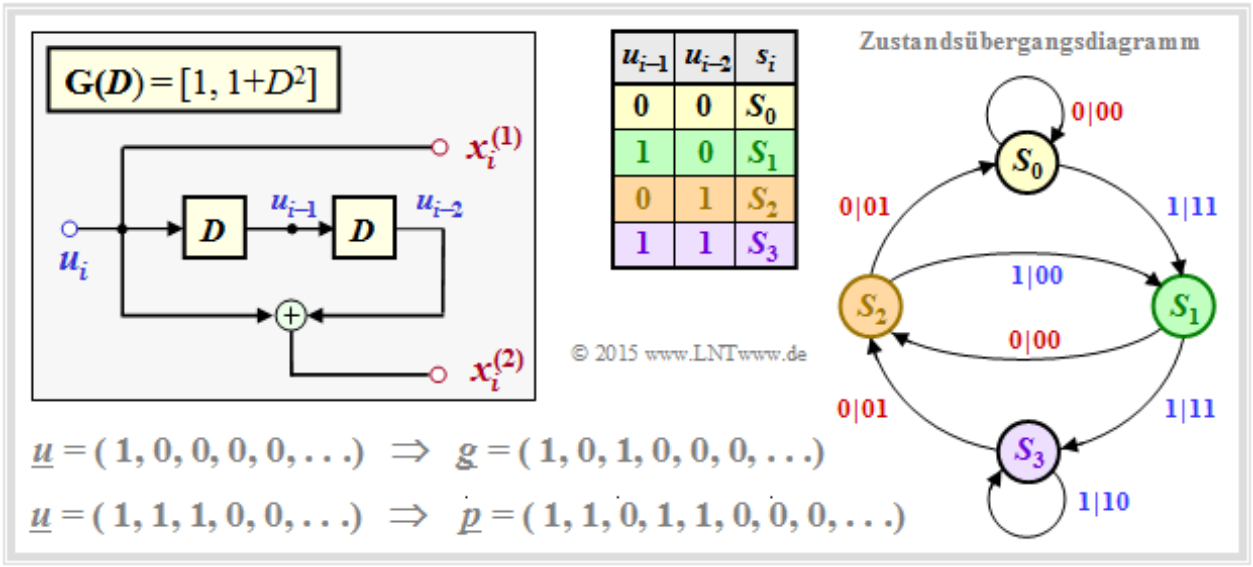
(2) Die Paritysequenzen ergeben sich gemäß $G_1(D) = G_2(D) = 1 + D^2$ ⇒ siehe **Aufgabe A4.8.**



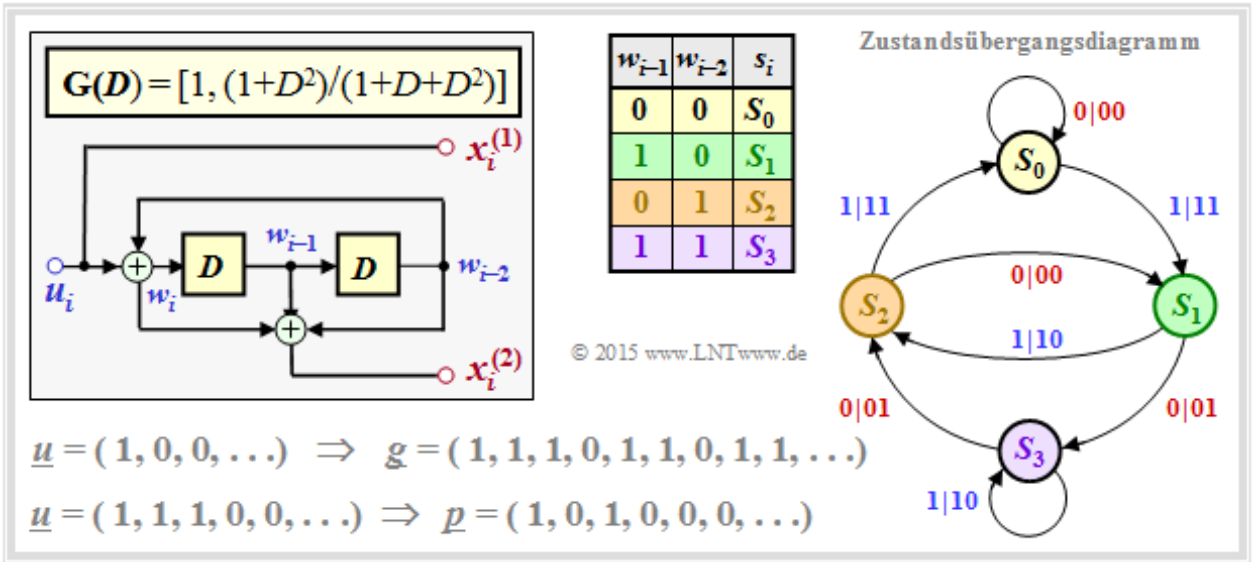
Erste Voraussetzung für Turbocodes: Rekursive Komponentencodes

Nichtrekursive Übertragungsfunktionen zur Erzeugung der Paritysequenzen bewirken einen Turbocode mit unzureichend kleiner Minimaldistanz. Grund für diese Unzulänglichkeit ist die endliche Impulsantwort $\underline{g} = (1, g_2, \text{...} , g_m, 0, 0, \text{...})$ mit $g_2, \text{...} , g_m \in \{0, 1\}$. Hierbei bezeichnet m das Codegedächtnis.

Die Grafik zeigt das Zustandsübergangsdiagramm für das Beispiel $\mathbf{G}(D) = [1, 1 + D^2]$. Die Übergänge sind mit „$u_i \mid u_i\, p_i$" beschriftet. Die Abfolge $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0 \rightarrow S_0 \rightarrow S_0 \rightarrow \text{...}$ führt bezüglich des Eingangs zur Informationssequenz $\underline{u} = (1, 0, 0, 0, 0, \text{...})$ und bezüglich des jeweils zweiten Codesymbols zur Paritysequenz $\underline{p} = (1, 0, 1, 0, 0, \text{...}) \Rightarrow$ identisch mit der Impulsantwort $\underline{g} \Rightarrow$ Gedächtnis $m = 2$.



Die untere Grafik gilt für einen sog. RSC–Code (*Recursive Systematic Convolutional*) entsprechend $\mathbf{G}(D) = [1, (1 + D^2)/(1 + D + D^2)]$. Hier führt die Folge $S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow \text{...}$ zur Impulsantwort $\underline{g} = (1, 1, 1, 0, 1, 1, 0, 1, 1, \text{...})$. Diese Impulsantwort setzt sich aufgrund der Schleife $S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1$ bis ins Unendliche fort. Dies ermöglicht bzw. erleichtert die iterative Decodierung.



Mehr Details zu den Beispielen auf dieser Seite finden Sie in **Aufgabe A4.8** und **Aufgabe A4.9.**

Zweite Voraussetzung für Turbocodes: Interleaving

Es ist offensichtlich, dass bei $G_1(D) = G_2(D)$ ein Interleaver (Π) unerlässlich ist. Ein weiterer Grund ist, dass die Apriori–Information als unabhängig vorausgesetzt wird. Somit sollten benachbarte (und somit möglicherweise stark abhängige) Bits für den jeweils anderen Teilcode weit auseinander liegen.

Für jeden RSC–Code ⇒ unendliche Impulsantwort $\underline{g}$ ⇒ gebrochen–rationale Übertragungsfunktion $G(D)$ gibt es nämlich gewisse Eingangssequenzen $\underline{u}$, die zu sehr kurzen Paritysequenzen $\underline{p} = \underline{u} * \underline{g}$ mit geringem Hamming–Gewicht $w_H(\underline{p})$ führen. Eine solche Sequenz ist beispielsweise in der unteren Grafik auf der **letzten Seite** angegeben: $\underline{u} = (1, 1, 1, 0, 0, ...)$. Dann gilt für die Ausgangssequenz:

$$P(D) = U(D) \cdot G(D) = (1 + D + D^2) \cdot \frac{1 + D^2}{1 + D + D^2} = 1 + D^2$$
$$\Rightarrow \quad \underline{p} = (1, 0, 1, 0, 0, ...).$$

Durch Interleaving (deutsch: *Verwürfelung*) wird nun mit großer Wahrscheinlichkeit sichergestellt, dass diese Sequenz $\underline{u} = (1, 1, 1, 0, 0, ...)$ in eine Sequenz $\underline{u}_\pi$ gewandelt wird,

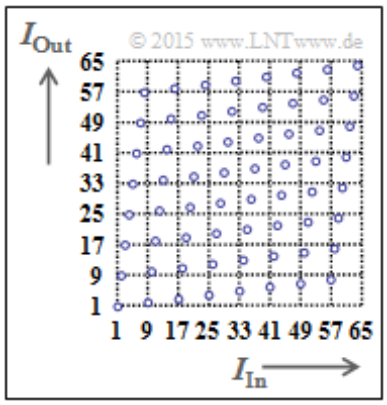
- die zwar ebenfalls nur drei Einsen beinhaltet,
- deren Ausgangssequenz aber durch ein großes Hamming–Gewicht $w_H(\underline{p})$ gekennzeichnet ist.

Somit gelingt es dem Decoder, solche „Problemsequenzen" iterativ aufzulösen.

Zur Beschreibung der Interleaver verwendet man die Zuordnung $I_{In} \rightarrow I_{Out}$. Diese Bezeichnungen stehen für die Indizes von Ausgangs– bzw. Eingangsfolge. Die Interleavergröße wird mit I_{max} benannt.

Es gibt mehrere, grundsätzlich verschiedene Interleaver–Konzepte:

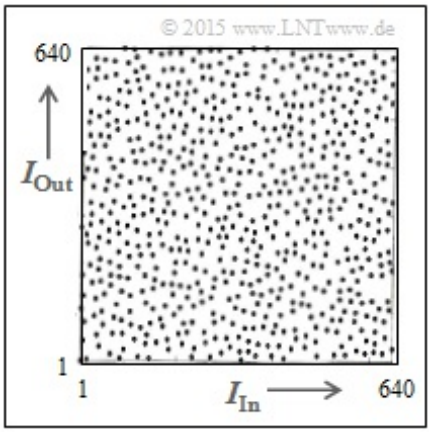
Bei einem **Block–Interleaver** füllt man eine Matrix mit S Spalten und Z Zeilen spaltenweise und liest die Matrix zeilenweise aus. Damit wird ein Informationsblock mit $I_{max} = S \cdot Z$ Bit deterministisch verwürfelt.



Die Grafik verdeutlicht das Prinzip für $I_{max} = 64$ ⇒ $1 \le I_{In} < 65$ und $1 \le I_{Out} < 65$. Die Reihenfolge der Ausgangsbits lautet dann:

1, 9, 17, 25, 33, 41, 49, 57, 2, 10, 18, ... , 48, 56 , 64.

Mehr Informationen zum Block–Interleaving gibt es in **Aufgabe Z4.8.**

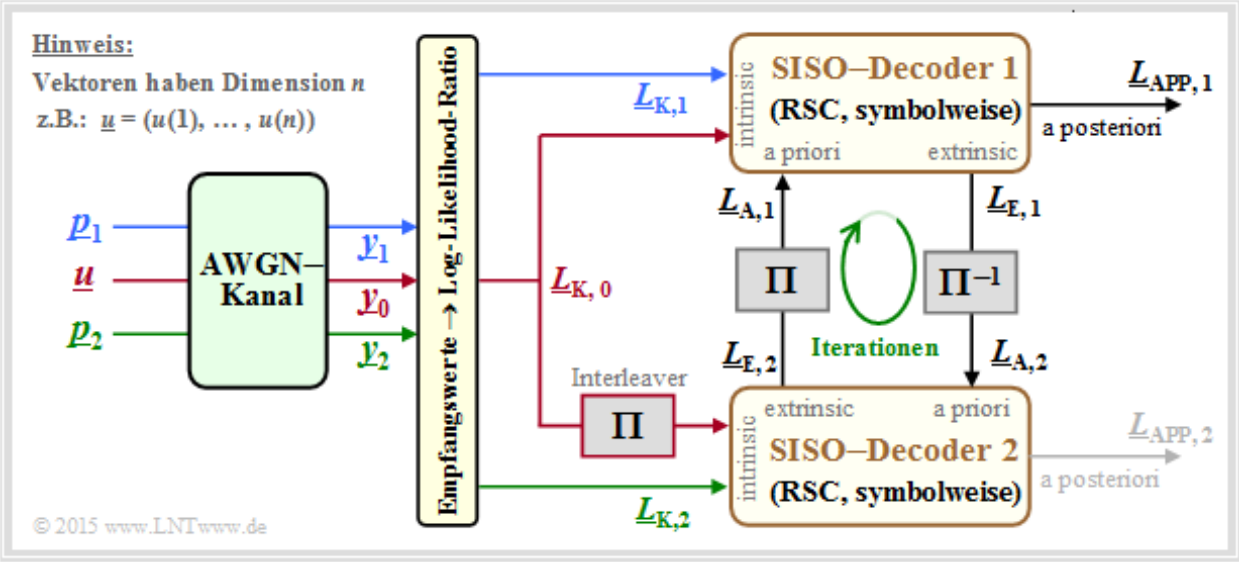


Turbocodes verwenden oft den ***S*–Random–Interleaver**. Dieser pseudozufällige Algorithmus mit dem Parameter „S" garantiert, dass zwei am Eingang weniger als S auseinander liegende Indizes am Ausgang mindestens im Abstand $S + 1$ auftreten. Die linke Grafik zeigt die S–Random–Kennlinie $I_{Out}(I_{In})$ für $I_{max} = 640$.

Auch dieser Algorithmus ist deterministisch, und man kann die Verwürfelung im Decoder rückgängig machen ⇒ *De–Interleaving*. Die Verteilung wirkt trotzdem „zufälliger" als bei Block–Interleaving.

Symbolweise iterative Decodierung eines Turbocodes

Die Decodierung eines Turbocodes geschieht grundsätzlich wie in **Kapitel 4.1** beschrieben. Aus der folgenden Grafik erkennt man aber einige nur für den Turbodecoder zutreffende Besonderheiten.



Vorausgesetzt ist ein Rate–1/3–Turbocode entsprechend der Beschreibung auf der **ersten Seite** dieses Abschnitts. Auch die Farbgebung für die Informationssequenz $\underline{u}$ und die beiden Paritysequenzen $\underline{p}_1$ und $\underline{p}_2$ sind an die früheren Grafiken angepasst. Weiter ist zu bemerken:

- Die Empfangsvektoren $\underline{y}_0$, $\underline{y}_1$ und $\underline{y}_2$ sind reellwertig und liefern die jeweilige Soft–Information bezüglich der Sequenzen $\underline{u}$ (Information), $\underline{p}_1$ (Parity für Coder 1) und $\underline{p}_2$ (Parity für Coder 2).
- Der Decoder 1 erhält die erforderliche intrinsische Information in Form der L–Werte $L_{\rm K,0}$ (aus $\underline{y}_0$) und $L_{\rm K,1}$ (aus $\underline{y}_1$) über jedes einzelne Bit der Sequenzen $\underline{u}$ und $\underline{p}_1$.
- Beim zweiten Decoder ist auch die Verwürfelung der Informationssequenz $\underline{u}$ durch den Interleaver zu berücksichtigen. Die zu verarbeitenden L–Werte sind somit $\pi(L_{\rm K,0})$ und $L_{\rm K,2}$.
- Beim allgemeinen **SISO–Decoder** am Ende von Kapitel 4.1 wurde der Informationsaustausch zwischen den beiden Komponentendecodern mit $\underline{L}_{\rm A,2} = \underline{L}_{\rm E,1}$ sowie $\underline{L}_{\rm A,1} = \underline{L}_{\rm E,2}$ angegeben. Ausgeschrieben auf Bitebene bedeuten diese Gleichungen mit $1 \le i \le n$:

$$L_{\rm A,2}(i) = L_{\rm E,1}(i) \quad \text{bzw.} \quad L_{\rm A,1}(i) = L_{\rm E,2}(i).$$

- Beim Turbodecoder muss bei diesem Informationsaustausch auch der Interleaver berücksichtigt werden. Dann gilt wieder für $i = 1, ..., n$:

$$L_{\rm A,2}\left(\pi(i)\right) = L_{\rm E,1}(i) \quad \text{bzw.} \quad L_{\rm A,1}(i) = L_{\rm E,2}\left(\pi(i)\right).$$

- Der Aposteriori–L–Wert wird in obigem Modell (willkürlich) vom Decoder 1 abgegeben. Dies lässt sich damit begründen, dass eine Iteration für einen zweifachen Informationsaustausch steht.

Leistungsfähigkeit der Turbocodes (1)

Wir betrachten wieder wie auf den letzten Seiten den Rate–1/3–Turbocode

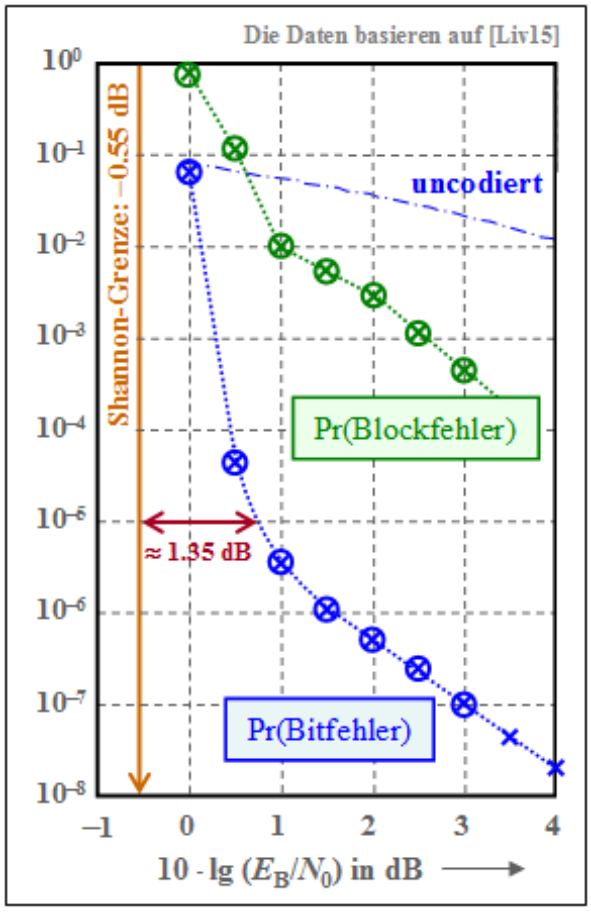
- mit gleichen Filterfunktionen $G_1(D) = G_2(D) = (1 + D^2)/(1 + D + D^2)$ ⇒ Gedächtnis $m = 2$,
- mit der Interleavergröße K; zunächst gelte $K = 10000$,
- eine ausreichende Anzahl an Iterationen ($I = 20$), hier nahezu gleichzusetzen mit „$I \to \infty$".

Die beiden RSC–Komponentencodes sind jeweils auf K Bit terminiert. Deshalb gruppieren wir

- die Informationssequenz $\underline{u}$ zu Blöcken mit je K Informationsbits, und
- die Codesequenz $\underline{x}$ zu Blöcken mit je $N = 3 \cdot K$ Codebits.

Die Grafik zeigt als grüne Kurve in doppelt–logarithmischer Darstellung die Blockfehlerwahrscheinlichkeit ⇒ Pr(Blockfehler) beim **AWGN–Kanal** in Abhängigkeit der Kenngröße $10 \cdot \lg(E_B/N_0)$. Man erkennt:

- Die Daten entstammen dem Vorlesungsskript **[Liv15].** Die mit Kreuzen markierten Punkte ergaben sich aus den Gewichtsfunktionen des Turbocodes mit Hilfe der **Union Bound.**
- Simulationsergebnisse aus [Liv15] sind in der Grafik durch Kreise markiert. Sie sind nahezu deckungsgleich mit den berechneten Werten.
- Die Union Bound ist nur eine obere Schranke, basierend auf ML–Decodierung. Der iterative Decoder ist suboptimal (also schlecher als ML). Diese beiden Effekte heben sich scheinbar auf.
- Etwa bei $10 \cdot \lg(E_B/N_0) = 1$ dB ist ein Knick im (grünen) Kurvenverlauf festzustellen, der mit der Steigungsänderung von Pr(Bitfehler) ⇒ blaue Kurve korrespondiert. Die Erklärung folgt unten.



Die blauen Kreuze (Berechnung) und die blauen Kreise (Simulation) bezeichnen die Bitfehlerwahrscheinlichkeit. Als Vergleichskurve ist die (strichpunktierte) Kurve für uncodierte Übertragung eingezeichnet. Anzumerken ist:

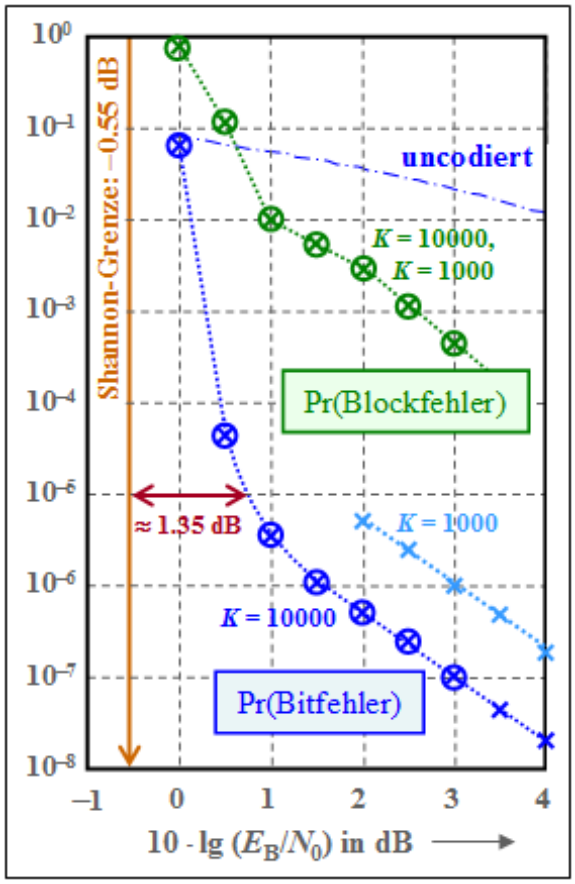
- Bei kleinen Abszissenwerten ist der Kurvenabfall in der gewählten Darstellung nahezu linear und ausreichend steil. Für Pr(Bitfehler) = 10^{-5} benötigt man $10 \cdot \lg(E_B/N_0) \approx 0.8$ dB.
- Im Vergleich zur **Shannon–Grenze,** die sich für die Coderate $R = 1/3$ zu $10 \cdot \lg(E_B/N_0) \approx -0.55$ dB ergibt, liegt unser Standard–Turbocode (mit Gedächtnis $m = 2$) nur etwa 1.35 dB entfernt.
- Ab $10 \cdot \lg(E_B/N_0) \approx 0.5$ dB verläuft die Kurve flacher. Ab ca. 1.5 dB ist der Verlauf wieder (fast) linear mit geringerer Steigung. Für Pr(Bitfehler) = 10^{-7} benötigt man etwa $10 \cdot \lg(E_B/N_0) = 3$ dB.

Die Bildbeschreibung wird auf der nächsten Seite fortgesetzt.

Leistungsfähigkeit der Turbocodes (2)

Wir betrachten weiter die (blaue) Bitfehlerwahrscheinlichkeit für die Interleavergröße $K = 10000$ und versuchen, den flacheren Abfall bei größerem E_B/N_0 zu erklären. Man spricht von einem *Error Floor*.

- Der Grund für dieses asymptotisch schlechtere Verhalten bei besserem Kanal (im Beispiel: ab ca. $10 \cdot \lg E_B/N_0 > 2$ dB) ist die Periode P der Coderimpulsantwort $\underline{g}$, wie auf der Seite **Zweite Voraussetzung: Interleaving** nachgewiesen.
- Im Beispiel ($m = 2$) ist die Periode $P = 2^m = 3$. Dadurch ist für $\underline{u} = (1, 1, 1) \Rightarrow w_H(\underline{u}) = 3$ trotz unbegrenzter Impulsantwort $\underline{g}$ die Paritysequenz begrenzt: $\underline{p} = (1, 0, 1) \Rightarrow w_H(\underline{p}) = 2$.
- Die Sequenz $\underline{u} = (0, ..., 0, 1, 0, 0, 1, 0, ...) \Rightarrow U(D) = D^x \cdot (1 + D^P)$ führt ebenfalls zu einem kleinen Hamming–Gewicht $w_H(\underline{p})$ am Ausgang, was den iterativen Decodierprozess erschwert.
- Eine gewisse Abhilfe schafft der Interleaver, der dafür sorgt, dass nicht die beiden Sequenzen $\underline{p}_1$ und $\underline{p}_2$ gleichzeitig durch sehr kleine Hamming–Gewichte $w_H(\underline{p}_1)$ und $w_H(\underline{p}_2)$ belastet sind.



- Aus der Grafik erkennt man, dass Pr(Bitfehler) umgekehrt proportional zur Interleavergröße K ist. Das heißt: Bei großem K funktioniert die Entspreizung ungünstiger Eingangssequenzen besser.
- Allerdings gilt die Näherung $K \cdot$ Pr(Bitfehler) = const. nur für größere E_B/N_0–Werte $\Rightarrow$ kleinere Bitfehlerwahrscheinlichkeiten. Der oben beschriebene Effekt tritt zwar auch bei kleinerem E_B/N_0 auf, doch sind dann die Auswirkungen auf die Bitfehlerwahrscheinlichkeit geringer.

Dagegen gilt der flachere Verlauf der Blockfehlerwahrscheinlichkeit (grüne Kurve weitgehend unabhängig von der Interleavergröße K, also sowohl für $K = 1000$ als auch für $K = 10000$. Im Bereich ab $10 \cdot \lg E_B/N_0 > 2$ dB dominieren aufgrund der kleinen Bitfehlerwahrscheinlichkeiten ($< 10^{-5}$) nämlich Einzelfehler, so dass hier die Näherung Pr(Blockfehler) ≈ Pr(Bitfehler) · K gültig ist.

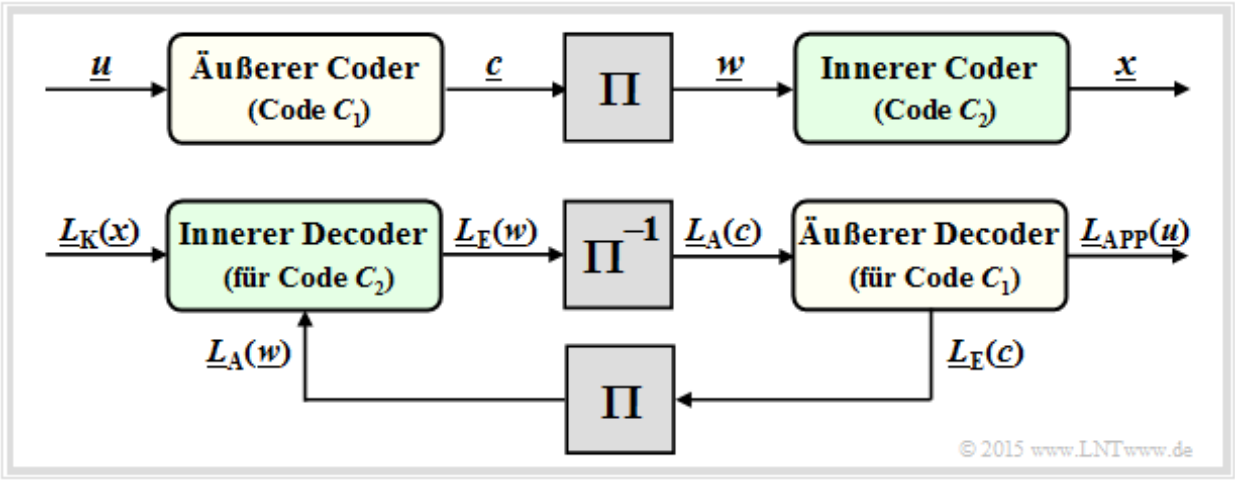
Die hier beispielhaft gezeigten Kurven für Bit– und Blockfehlerwahrscheinlichkeit gelten qualitativ auch für $m > 2$, zum Beispiel für den Turbocode von UMTS und LTE (jeweils $m = 3$), der in **Aufgabe A4.10** analysiert wird. Es ergeben sich aber einige quantitative Unterschiede:

- Die Kurve verläuft bei kleinem E_B/N_0 steiler und der Abstand von der Shannongrenze ist etwas geringer als im hier gezeigten Beispiel für $m = 2$.
- Auch für größeres m gibt es einen *Error Floor*. Der Knick in den dargestellten Kurven erfolgt aber später, also bei kleineren Fehlerwahrscheinlichkeiten.

Seriell verkettete Turbocodes – SCCC

Die bisher betrachteten Turbocodes werden manchmal auch als *Parallel Concatenated Convolutional Codes* (PCCC) bezeichnet. Einige Jahre nach Berrou's Erfindung wurden von anderen Autoren auch *Serial Concatenated Convolutional Codes* (SCCC) entsprechend folgender Grafik vorgeschlagen.

- Die Informationssequenz $\underline{u}$ liegt am äußeren Faltungscoder C_1 an. Dessen Ausgangssequenz sei $\underline{c}$.
- Nach dem Interleaver (Π) folgt der innere Faltungscoder C_2. Die Codesequenz wird $\underline{x}$ genannt.
- Die resultierende Coderate ist $R = R_1 \cdot R_2$. Bei Rate–1/2–Komponentencodes ist $R = 1/4$.



Die untere Grafik zeigt den SCCC–Decoder und verdeutlicht die Verarbeitung der L–Werte und den Austausch der extrinsischen Information zwischen den beiden Komponentencoder:

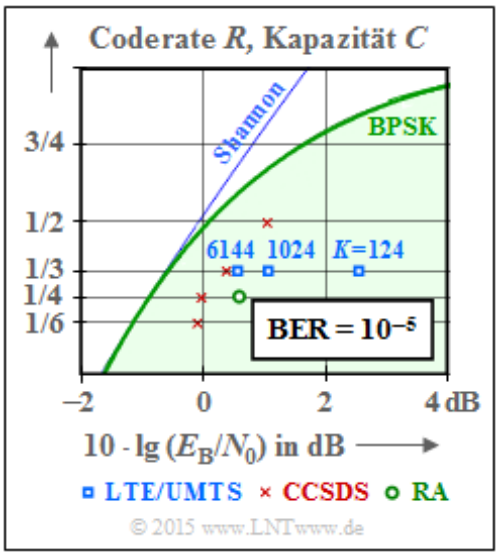
- Der innere Decoder (für den Code C_2) erhält vom Kanal die intrinsische Information $\underline{L}_{\rm K}(\underline{x})$ und vom äußeren Decoder (nach Interleaving) die Apriori–Information $\underline{L}_{\rm A}(\underline{w})$ mit $\underline{w} = \pi(\underline{c})$. An den äußeren Decoder wird die extrinsische Information $\underline{L}_{\rm E}(\underline{w})$ abgegeben.
- Der äußere Decoder (für C_1) verarbeitet die Apriori–Information $\underline{L}_{\rm A}(\underline{c})$, also die extrinsische Information $\underline{L}_{\rm E}(\underline{w})$ nach dem De–Interleaving. Er liefert die extrinsische Information $\underline{L}_{\rm E}(\underline{c})$.
- Nach hinreichend vielen Iterationen ergibt sich das das gewünschte Decodierergebnis in Form der Aposteriori–L–Werte $\underline{L}_{\rm APP}(\underline{u})$ der Informationssequenz $\underline{u}$.

Wichtig für seriell verkettete Faltungscodes ist, dass der innere Code rekursiv ist (also ein RSC–Code). Der äußere Code C_1 kann auch nichtrekursiv sein. Zur Leistungsfähigkeit solcher Codes ist anzumerken:

- SCCCs sind bei großem $E_{\rm B}/N_0$ besser als PCCCs $\Rightarrow$ niedrigerer *Error Floor*. Die Aussage gilt schon für SCCC–Komponentencodes mit Gedächtnis $m = 2$ (nur vier Trelliszustände), während bei PCCC das Gedächtnis $m = 3$ bzw. $m = 4$ (acht bzw. sechzehn Trelliszustände) sein sollte.
- Im unteren Bereich (kleines $E_{\rm B}/N_0$) ist dagegen der beste seriell verkettete Faltungscode (SCCC) um einige Zehntel Dezibel schlechter als der vergleichbare Turbocode gemäß Berrou (PCCC). Entsprechend größer ist auch der Abstand von der Shannongrenze.

Einige Anwendungsgebiete für Turbocodes

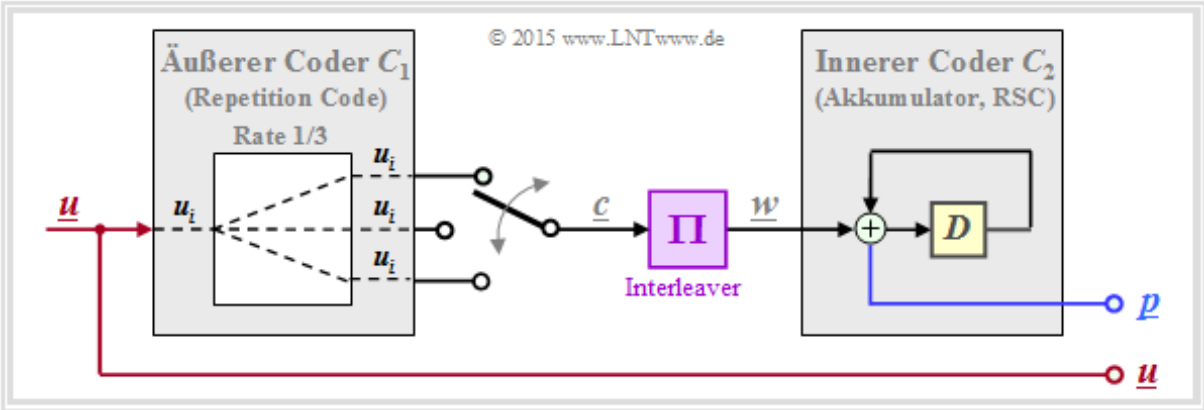
In fast allen neueren Kommunikationssystemen (nach 1993 standardisiert) werden Turbocodes eingesetzt. Die Grafik zeigt deren Leistungsfähigkeit beim AWGN–Kanal im Vergleich zur **Shannonschen Kanalkapazität** (blaue Kurve).



Der grün hinterlegte Bereich „BPSK” gibt die Shannongrenze für Nachrichtensystemee mit binärem Eingang an, mit der nach dem **Kanalcodierungstheorem** eine fehlerfreie Übertragung gerade noch möglich ist.

Anzumerken ist, dass hier für die eingezeichneten Kanalcodes von standardisierten Systemen die Fehlerrate 10^{-5} zugrunde liegt, während die informationstheoretischen Kapazitätskurven (Shannon, BPSK) für die Fehlerwahrscheinlichkeit 0 gelten.

- Die blauen Rechtecke markieren die Turbocodes für **UMTS.** Diese sind Rate–1/3–Codes mit Gedächtnis $m = 3$. Die Leistungsfähigkeit hängt stark von der Interleavergröße ab. Mit $K = 6144$ liegt dieser Code nur etwa 1 dB rechts von der Shannon–Grenze. **LTE** verwendet die gleichen Turbocodes. Geringfügige Unterschiede ergeben sich aufgrund des unterschiedlichen Interleavers.
- Die roten Kreuze markieren die Turbocodes nach CCSDS (*Consultative Comittee for Space Data Systems*), entwickelt für den Einsatz bei fernen Weltraummissionen. Diese Klasse geht von der einheitlichen Interleavergröße $K = 6920$ aus und stellt Codes der Rate 1/6, 1/4, 1/3 und 1/2 zur Verfügung. Die niedrigsten Coderaten erlauben einen Betrieb mit $10 \cdot \lg (E_B/N_0) \approx 0$ dB.
- Der grüne Kreis steht für einen sehr einfachen *Repeat–Accumulate* (RA) Code, einem seriell–verketteten Turbocode. Der äußere Decoder verwendet einen **Wiederholungscode** (englisch: *Repetition Code*), im gezeichneten Beispiel mit der Rate $R = 1/3$. Nach dem Interleaver folgt ein RSC–Code mit $G(D) = 1/(1 + D)$ $\Rightarrow$ Gedächtnis $m = 1$. Bei systematischer Ausführung ist die Gesamtcoderate $R = 1/4$ (zu jedem Informationsbit noch drei Paritybits). Aus der oberen Grafik erkennt man, dass dieser einfache RA–Code überraschend gut ist. Mit der Interleavergröße $K = 300000$ beträgt der Abstand von der Shannon–Grenze lediglich ca. 1.5 dB.



In der oberen Grafik nicht eingetragen sind die Turbocodes für den Standard *DVB Return Channel Terrestrial* (RCS) sowie für den WiMax–Standard (IEEE 802.16), die ähnliche Turbocodes benutzen.

Einige Charakteristika der LDPC–Codes (1)

Die *Low–density Parity–check Codes* – kurz LDPC–Codes – wurden bereits Anfang der 1960er Jahre erfunden und gehen auf die Dissertation **[Gal63]** von Robert G. Gallager zurück.

Die Idee kam allerdings aufgrund der damaligen Prozessorentechnologie um einige Jahrzehnte zu früh. Schon drei Jahre nach Berrou's Erfindung der Turbocodes 1993 erkannten dann allerdings David J. C. MacKay und Radford M. Neal das riesige Potential der LDPC–Codes, wenn man diese ebenso wie die Turbocodes iterativ symbolweise decodiert. Sie erfanden die LDPC–Codes quasi neu.

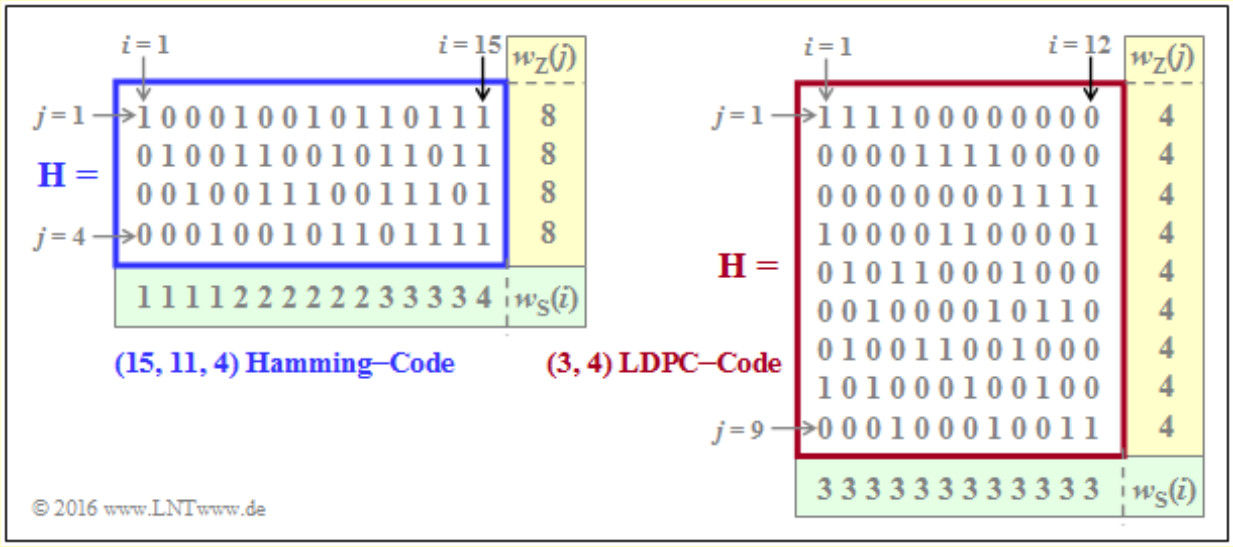
Wie aus dem Namensbestandteil „Parity–check" bereits hervorgeht, handelt es sich bei diesen Codes um lineare Blockcodes entsprechend den Ausführungen in **Kapitel 1.** Deshalb gilt auch hier:

- Das Codewort ergibt sich aus dem Informationswort $\underline{u}$ (dargestellt mit k Binärsymbolen) und der **Generatormatrix G** der Dimension $k \times n$ zu $\underline{x} = \underline{u} \cdot \mathbf{G} \Rightarrow$ Codewortlänge n.
- Die Prüfgleichungen ergeben sich aus der Identität $\underline{x} \cdot \mathbf{H}^{\rm T} \equiv 0$, wobei **H** die Prüfmatrix bezeichnet. Diese besteht aus m Zeilen und n Spalten. Während im ersten Kapitel grundsätzlich $m = n - k$ gegolten hat, fordern wir für die LPDC–Codes lediglich noch $m \ge n - k$.

Ein gravierender Unterschied zwischen einem LDPC–Code und einem herkömmlichen Blockcode nach der Beschreibung im ersten Kapitel ist, dass die Prüfmatrix **H** nur spärlich mit Einsen besetzt ist.

Beispiel: Die Grafik zeigt beispielhaft die Prüfmatrizen **H** für

- den Hamming–Code mit Codelänge $n = 15$, $m = 4 \Rightarrow k = 11$ Informationsbits,
- den LDPC–Code aus **[Liv15]** der Länge $n = 12$ und mit $m = 9$ Prüfgleichungen $\Rightarrow k \ge 3$.

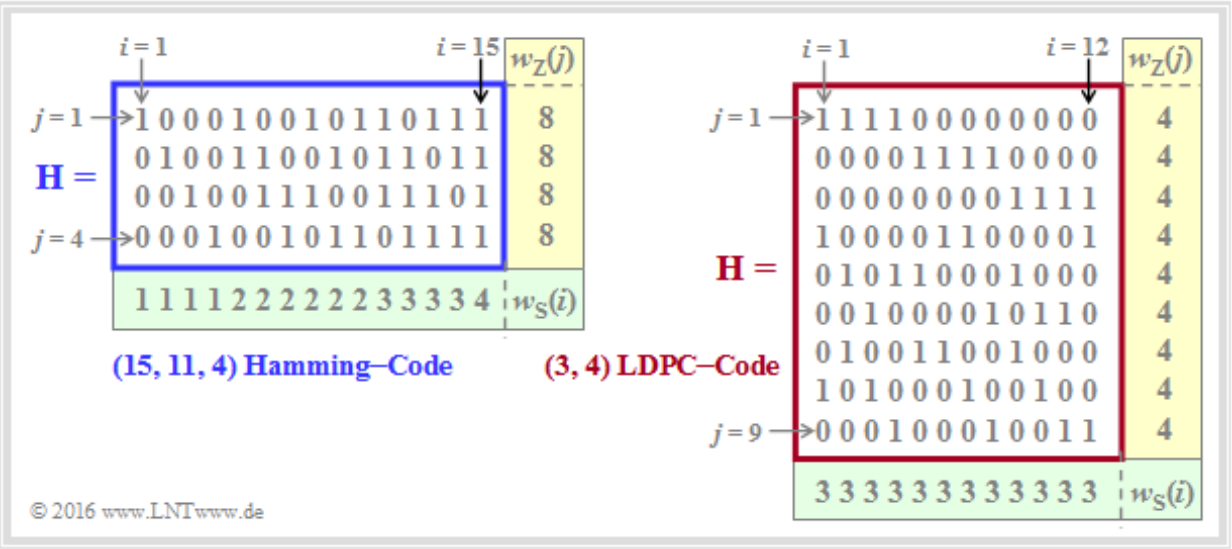


In der linken Grafik beträgt der Anteil der Einsen 32/60 ≈ 53.3%, wohingegen in der rechten Grafik der Einsen–Anteil mit 36/108 = 33.3% geringer ist. Bei den für die Praxis relevanten LDPC–Codes (großer Länge) ist der Einsen–Anteil noch deutlich niedriger.

Hinweis: Auf der nächsten Seite wird auf diese Grafik noch mehrmals Bezug genommen.

Einige Charakteristika der LDPC–Codes (2)

Wir analysieren nun die beiden Prüfmatrizen anhand der Rate und des Hamming–Gewichts.



- Die Rate des betrachteten Hamming–Codes (linke Grafik) ist $R = k/n = 11/15 \approx 0.733$. Das Hamming–Gewicht einer jeden der vier Zeilen ist $w_Z = 8$, während die Hamming–Gewichte $w_S(i)$ der Spalten zwischen 1 und 4 variieren. Für die Spalten–Laufvariable gilt hier: $1 \le i \le 15$.
- Beim betrachteten LDPC–Code gibt es in allen Zeilen vier Einsen $\Rightarrow$ $w_Z = 4$ und in allen Spalten drei Einsen $\Rightarrow w_S = 3$. Die Codebezeichnung **(w_Z, w_S) LDPC–Code** verwendet genau diese Parameter. Beachten Sie die unterschiedliche Nomenklatur zum „(n, k, m) Hamming–Code”.
- Man spricht hier von einem **regulären LDPC–Code**, da alle Zeilengewichte $w_Z(j)$ für $1 \le j \le m$ konstant gleich w_Z sind und auch alle Spaltengewichte (mit den Indizes $1 \le i \le n$) gleich sind: $w_S(i) = w_S = \text{const}$. Ist diese Bedingung nicht erfüllt, so liegt ein *irregulärer LDPC–Code* vor.
- Für die Coderate kann man allgemein (also, wenn k nicht bekannt ist) nur eine Schranke angeben: $\boldsymbol{R \ge 1 - w_S/w_Z}$. Das Gleichheitszeichen gilt dann, wenn alle Zeilen von $\mathbf{H}$ linear unabhängig sind $\Rightarrow$ $m = n - k$. Dann ergibt sich die herkömmliche Gleichung: $R = 1 - w_S/w_Z = 1 - m/n = k/n$.
- Dagegen gilt für die Coderate eines irregulären LDPC–Codes und auch für den links skizzierten (15, 11, 4) Hammingcode:

$$R \ge 1 - \frac{\mathrm{E}[w_S]}{\mathrm{E}[w_Z]} \quad \text{mit} \quad \mathrm{E}[w_S] = \frac{1}{n} \cdot \sum_{i=1}^{n} w_S(i) \quad \text{und} \quad \mathrm{E}[w_Z] = \frac{1}{m} \cdot \sum_{j=1}^{m} w_Z(j).$$

Da beim Hamming–Code die $m = n - k$ Prüfgleichungen linear voneinander unabhängig sind, kann das „$\ge$”–Zeichen durch das Gleichheitszeichen ersetzt werden, was gleichzeitig $R = k/n$ bedeutet.

Weitere Informationen zu diesem Thema finden Sie in **Aufgabe A4.11** und in **Aufgabe Z4.11**

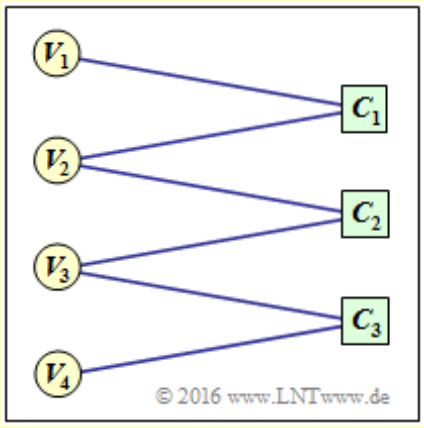
Zweiteilige LDPC–Graphenrepräsentation – Tanner–Graph (1)

Alle wesentlichen Merkmale eines LDPC–Codes sind in der Prüfmatrix $\mathbf{H} = (h_{j,i})$ enthalten und lassen sich durch einen so genannten **Tanner–Graphen** darstellen. Es handelt sich um eine *Bipartite Graph Representation*, wobei die deutsche Übersetzung von „bipartite" in etwa „zweiteilig" lautet. Bevor wir beispielhafte Tanner–Graphen genauer betrachten und analysieren, müssen zuerst noch einige geeignete Beschreibungsgrößen definiert werden:

- Die n Spalten der Prüfmatrix $\mathbf{H}$ stehen jeweils für ein Codewortbit. Da jedes Codewortbit sowohl ein Informationsbit als auch ein Prüfbit sein kann, hat sich hierfür die neutrale Bezeichnung **Varibale Node** (VN) durchgesetzt. Das i–te Codewortbit wird V_i genannt und die Menge aller *Variable Nodes* (VNs) ist $\{V_1, ..., V_i, ..., V_n\}$.
- Die m Zeilen von $\mathbf{H}$ beschreiben jeweils eine Prüfgleichung (englisch: *Parity–check Equation*). Wir bezeichnen im folgenden eine solche Prüfgleichung als **Check Node** (CN). Die Menge aller *Check Nodes* (CNs) ist $\{C_1, ..., C_j, ..., C_m\}$, wobei C_j die Prüfgleichung der j–ten Zeile angibt.
- Im Tanner–Graphen werden die n *Variable Nodes* V_i als Kreise und die m *Check Nodes* C_j als Quadrate dargestellt. Ist das Matrixelement in Zeile j und Spalte i $\Rightarrow h_{j,i} = 1$, so gibt es eine Verbindungslinie (englisch: *Edge*) zwischen dem *Variable Node* V_i und dem *Check Node* C_j.

Beispiel: Sie sehen rechts einen beispielhaften Tannergraphen zur Verdeutlichung obiger Begriffe mit

- den *Variable Nodes* (kurz: VNs) V_1 bis V_4, und
- den *Check Nodes* (kurz: CNs) C_1 bis C_3.



Der zugehörige Code hat allerdings keinerlei praktische Bedeutung. Man erkennt aus der Grafik:

- Die Codelänge ist $n = 4$ und es gibt $m = 3$ Prüfgleichungen. Damit hat die Prüfmatrix $\mathbf{H}$ die Dimension 3×4.
- Es gibt insgesamt sechs Verbindungslinien (englisch: *Edges*). Damit sind sechs der zwölf Elemente $h_{j,i}$ von $\mathbf{H}$ Einsen.
- Bei jedem *Check Node* kommen zwei Linien an. Das bedeutet, dass die Hamming–Gewichte aller Zeilen gleich sind: $w_S(j) = 2 = w_S$.
- Von den Knoten V_1 und V_4 gibt es jeweils nur einen Übergang zu einem *Check Node*, von V_2 und V_3 dagegen zwei. Aus diesem Grund handelt es sich um einen *irregulären Code*.

Die Prüfmatrix lautet demnach:

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

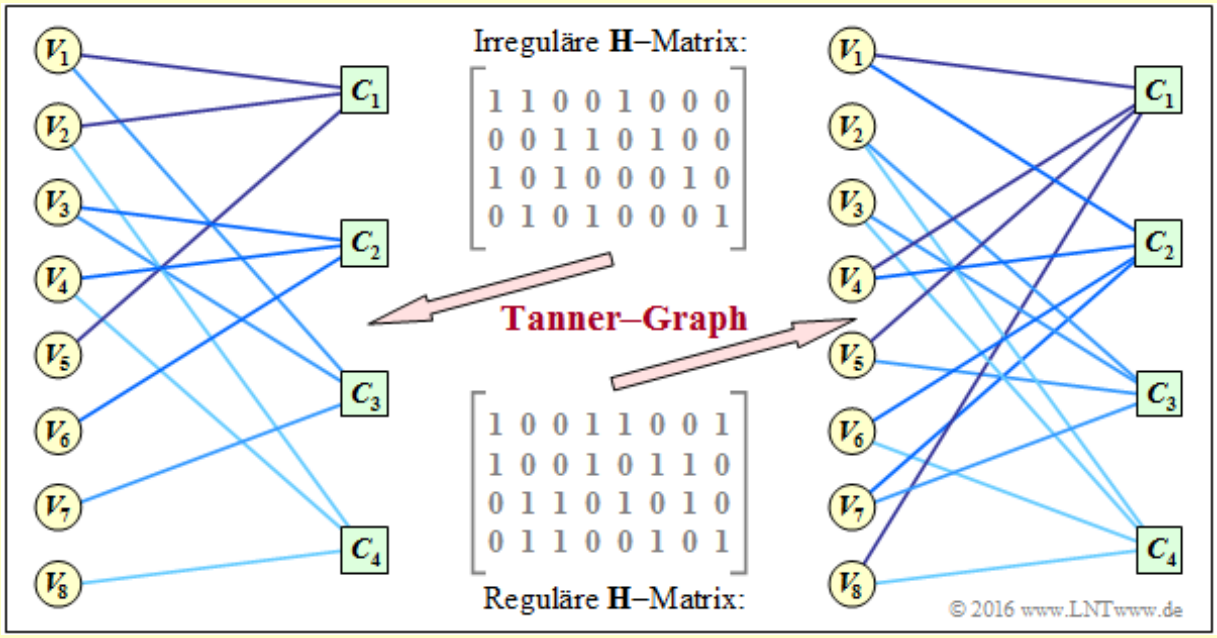
Auf der nächsten Seite folgt ein praxisrelevanteres Beispiel.

Zweiteilige LDPC–Graphenrepräsentation – Tanner–Graph (2)

Beispiel: In **Aufgabe A4.11** wurden zwei Prüfmatrizen analysiert:

- Der Coder entsprechend der Matrix $\mathbf{H}_1$ ist systematisch. Die Codeparameter sind $n = 8$, $k = 4$ und $m = 4 \Rightarrow$ Rate 1/2. Der Code ist irregulär, da die Hamming–Gewichte nicht für alle Spalten gleich sind. In der folgenden Grafik ist diese „irreguläre **H**–Matrix" oben angegeben.
- Unten angegeben ist die „reguläre **H**–Matrix" entsprechend der Matrix $\mathbf{H}_3$ in Aufgabe A4.11. Die Zeilen sind Linearkombinationen der oberen Matrix. Für diesen nicht systematischen Coder gilt mit $w_S = 2$ und $w_Z = 4$ ebenfalls:

$$R \geq 1 - \frac{w_S}{w_Z} = 1 - \frac{2}{4} = 1/2 \,.$$



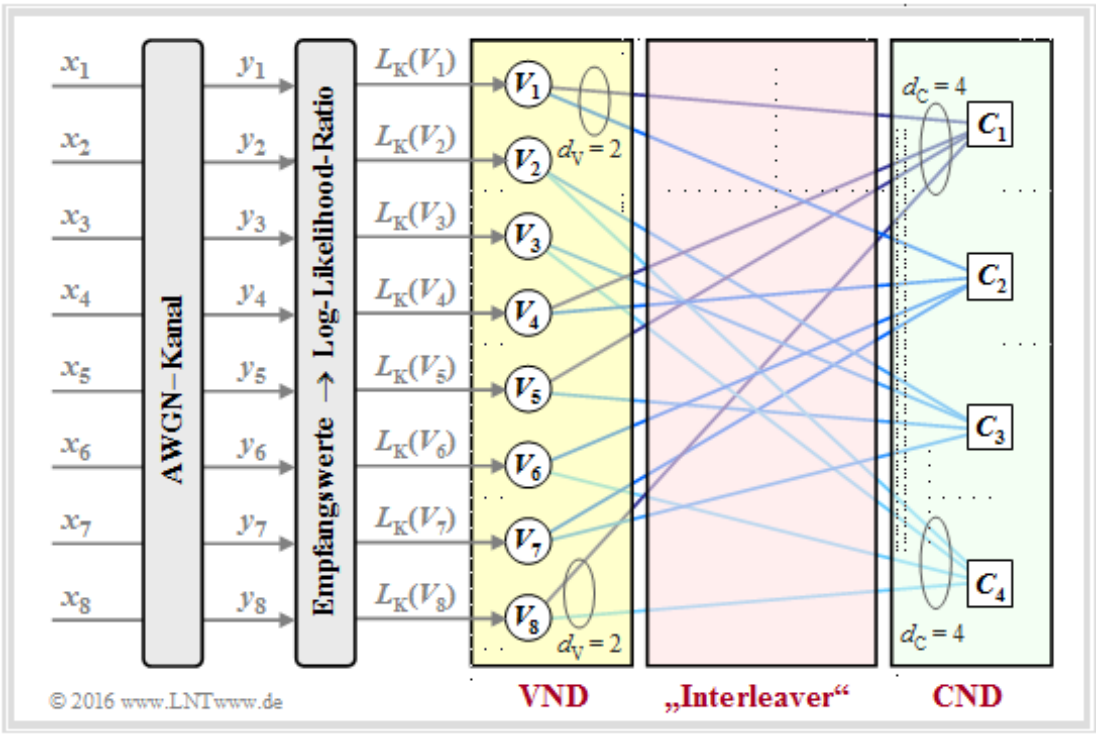
Die Grafik zeigt die zugehörigen Tanner–Graphen:

- Der linke Graph bezieht sich auf die irreguläre Matrix. Die acht *Variable Nodes* (abgekürzt VNs) V_i sind mit den vier *Check Nodes* (abgekürzt CNs) C_j verbunden, falls das Element in Zeile j und Spalte i $\Rightarrow$ $h_{j,i}$ gleich 1 ist. Andernfalls (falls $h_{j,i} = 0$) besteht keine Verbindung.
- Der links dargestellte Graph ist für die **iterative symbolweise Decodierung** nicht sonderlich gut geeignet. Die VNs V_5, ..., V_8 sind jeweils nur mit einem CN verbunden, was für die Decodierung keinerlei Information liefert.
- Im rechten Tanner–Graph für den regulären Code erkennt man, dass hier von jedem VN V_i zwei Verbindungslinien (englisch: *Edges*) abgehen und von jedem CN C_j deren vier. Damit ist bei der Decodierung in jeder Iterationsschleife ein Informationsgewinn möglich.
- Man erkennt zudem, dass hier beim Übergang vom irregulären zum äquivalenten regulären Code der Einsen–Anteil zunimmt, im Beispiel von 37.5% auf 50%. Diese Aussage kann allerdings nicht verallgemeinert werden.

Iterative Decodierung von LDPC–Codes (1)

Als Beispiel für die iterative LDPC–Decodierung wird nun der sog. **Message–passing Algorithm** beschrieben. Wir verdeutlichen diesen anhand des rechten Tanner–Graphen auf der **vorherigen Seite** und damit für die dort angegebene reguläre Prüfmatrix.

Bei diesem Decodieralgorithmus erfolgt abwechselnd (oder iterativ) ein Informationsaustausch zwischen den *Variable Nodes* (VNs) V_1, ... , V_n und den *Check Nodes* (CNs) C_1, ... , C_m.

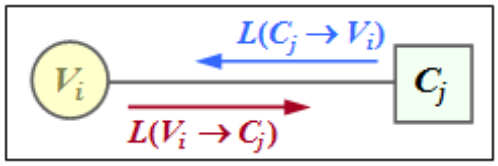


Für das betrachtete Beispiel gilt:

- Es gibt $n = 8$ VNs und $m = 4$ CNs. Da ein regulärer LDPC–Code vorliegt, gehen von jedem VN $d_V = 2$ Verbindungslinien zu einem CN und jeder CN ist mit $d_C = 4$ VNs verbunden.
- Der *Variable Node Degree* d_V ist gleich dem Hamming–Gewicht einer jeden Spalte (w_S) und für den *Check Node Degree* gilt: $d_C = w_Z$ (Hamming–Gewicht einer jeden Zeile).
- In der folgenden Beschreibung verwenden wir auch die Begriffe *Nachbarn eines VN* $\Rightarrow N(V_i)$ sowie *Nachbarn eines CN* $\Rightarrow N(C_j)$, wobei wir uns hier auf implizite Definitionen beschränken:

$$N(V_1) = \{C_1, C_2\}, \quad N(V_1) = \{C_3, C_4\}, \ \ldots \ , \quad N(V_8) = \{C_1, C_4\},$$
$$N(C_1) = \{V_1, V_4, V_5, V_8\}, \ \ldots \ , \quad N(C_4) = \{V_2, V_3, V_6, V_8\}.$$

Die Skizze aus **[Liv15]** zeigt den Austausch von Information zwischen dem *Variable Node* V_i und dem *Check Node* C_j – ausgedrückt durch **Log–Likelihood Ratios** (kurz: L–Werte). Der Informationsaustausch geschieht in zwei Richtungen:



- $L(V_i \rightarrow C_j)$: Extrinsische Information aus Sicht von V_i, Apriori–Information aus Sicht von C_j.
- $L(C_j \rightarrow V_i)$: Extrinsische Information aus Sicht von C_j, Apriori–Information aus Sicht von V_i.

Iterative Decodierung von LDPC–Codes (2)

Die Beschreibung des Decodieralgorithmus wird fortgesetzt:

Initialisierung: Zu Beginn der Decodierung erhalten die *Variable Nodes* (VNs) keine Apriori–Information von den *Check Nodes* (CNs), und es gilt für $1 \le i \le n$: $L(V_i \to C_j) = L_{\rm K}(V_i)$. Wie aus der letzten **Grafik** ersichtlich, ergeben sich diese Kanal–L–Werte $L_{\rm K}(V_i)$ aus den Empfangswerten y_i.

Check Node Decoder: Jeder Knoten C_j repräsentiert eine Prüfgleichung. So steht C_1 für „$V_1 + V_4 + V_5 + V_8 = 0$”. Man erkennt den Zusammenhang zur extrinsischen Information bei der symbolweisen Decodierung des *Single Parity–check Codes*. In Analogie zu **Kapitel 4.1** und **Aufgabe A4.4** gilt somit für den extrinsischen L–Wert von C_j und gleichzeitig für die Apriori–Information bezüglich V_i:

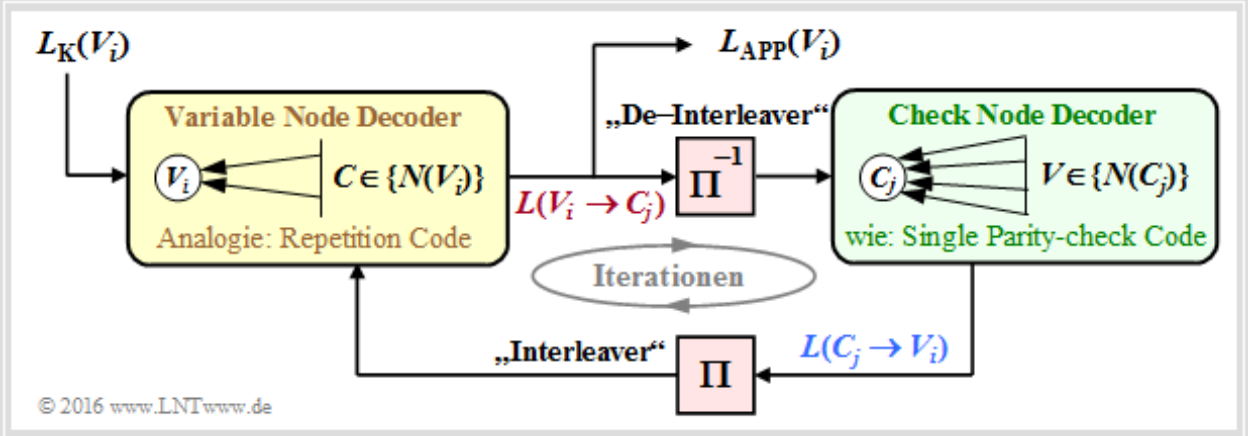
$$L(C_j \to V_i) = 2 \cdot \tanh^{-1}\left[\prod_{V \in N(C_j)\,,\,V \neq V_i} \tanh\left[L(V \to C_j)/2\right]\right].$$

Variable Node Decoder: Im Gegensatz zum *Check Node Decoder* (CND) besteht beim *Variable Node Decoder* (VND) eine Verwandtschaft zur Decodierung eines *Repetition Codes*, weil alle mit V_i verbundenen *Check Nodes* C_j demselben Bit entsprechen $\Rightarrow$ dieses Bit wird quasi $d_{\rm V}$ mal wiederholt.

In Analogie zu **Kapitel 4.1** gilt für den extrinsischen L–Wert von V_i und gleichzeitig für die Apriori–Information bezüglich C_j:

$$L(V_i \to C_j) = L_{\rm K}(V_i) + \sum_{C \in N(V_i)\,,\,C \neq C_j} L(C \to V_i)\,.$$

Das folgende Schaubild des beschriebenen Decodieralgorithmus' für LDPC–Codes zeigt Ähnlichkeiten mit der Vorgehensweise bei **seriell verketteten Turbocodes.** Um eine vollständige Analogie zwischen der LDPC– und der Turbodecodierung herzustellen, ist auch hier zwischen dem *Variable Node Decoder* (VND) und dem *Check Node Decoder* (CND) ein *Interleaver* sowie ein *De–Interleaver* eingezeichnet. Da es sich hierbei nicht um reale Systemkomponenten handelt, sondern nur um Analogien, haben wir diese Begriffe in Hochkommata gesetzt.

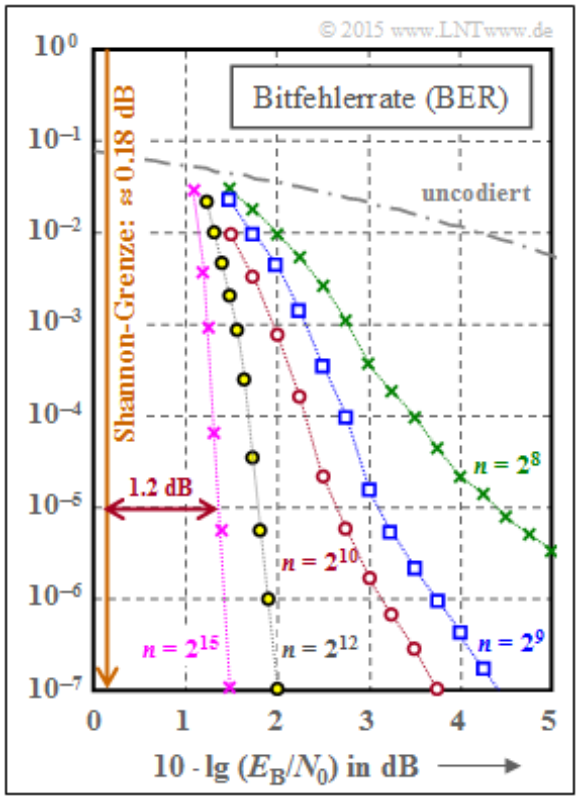


Leistungsfähigkeit der LDPC–Codes (1)

Wir betrachten nun wie in **[Str14]** fünf reguläre LDPC–Codes mit den folgenden Eigenschaften:

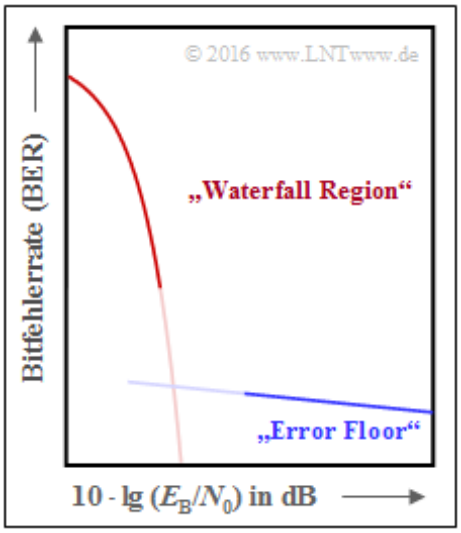
- Die Prüfmatrizen **H** weisen jeweils n Spalten und $m = n/2$ linear voneinander unabhängige Zeilen auf. In jeder Zeile gibt es $w_Z = 6$ Einsen und in jeder Spalte $w_S = 3$ Einsen.
- Der Einsen–Anteil beträgt $w_Z/n = w_S/m$, so dass bei großer Codewortlänge n die Klassifizierung „*Low–density*" gerechtfertigt ist. Für die rote Kurve ($n = 2^{10}$) ist der Einsen–Anteil 0.6%.
- Die Rate aller Codes beträgt $R = 1 - w_S/w_Z = 1/2$. Wegen der linearen Unabhängigkeit der Matrixzeilen gilt aber auch $R = k/n$ mit der Informationswortlänge $k = n - m = n/2$.

Die Grafik zeigt die Bitfehlerrate (BER) abhängig vom AWGN–Parameter $10 \cdot \lg E_B/N_0$. Zum Vergleich ist die Kurve für uncodierte Übertragung eingezeichnet.



Man erkennt:

- Die Bitfehlerrate ist um so kleiner, je länger der Code ist. Für $10 \cdot \lg E_B/N_0 = 2$ dB und $n = 2^8 = 256$ ergibt sich BER $\approx 10^{-2}$. Für $n = 2^{12} = 4096$ dagegen nur BER $\approx 2 \cdot 10^{-7}$.
- Mit $n = 2^{15} = 32768$ (violette Kurve) benötigt man $10 \cdot \lg E_B/N_0 \approx 1.35$ dB für BER $= 10^{-5}$. Der Abstand von der Shannon–Grenze (0.18 dB für $R = 1/2$ und BPSK) beträgt ca. 1.2 dB.



Die Kurven für $n = 2^8$ bis $n = 2^{10}$ weisen zudem auf einen Effekt hin, den wir schon bei den **Turbocodes** festgestellt haben: Zuerst fällt die BER–Kurve steil ab ⇒ „Waterfall Region", danach folgt ein Knick und ein Verlauf mit deutlich geringerer Steigung ⇒ „Error Floor". Die qualitative Grafik links verdeutlicht den Effekt, der natürlich nicht abrupt einsetzt (Übergang nicht eingezeichnet).

Ein (LDPC–)Code ist immer dann als gut zu bezeichnen, wenn

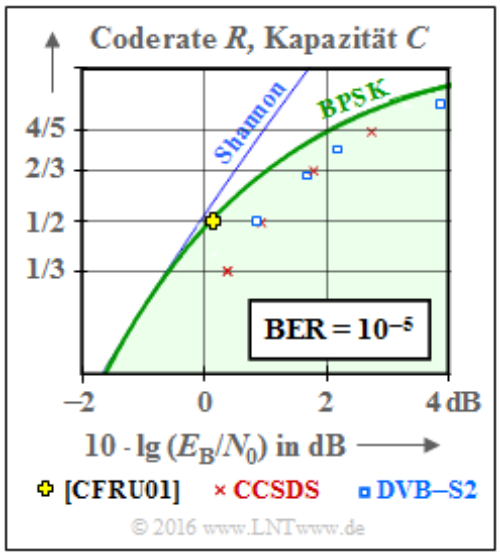
- die BER–Kurve nahe der Shannon–Grenze steil abfällt,
- der „Error Floor" (Ursachen hierfür siehe nächste Seite) bei sehr niedrigen BER–Werten liegt,
- die Anzahl der erforderlichen Iterationen handhabbar ist, und
- diese Eigenschaften nicht erst bei nicht mehr praktikablen Blocklängen erreicht werden.

Leistungsfähigkeit der LDPC–Codes (2)

In diesem Kapitel wurden vorwiegend reguläre LDPC–Codes behandelt, auch im BER–Diagramm auf der letzten Seite.

Die Ignoranz der irregulären LDPC–Codes ist nur der Kürze dieses Kapitels geschuldet, nicht deren Leistungsfähigkeit.

Im Gegenteil: Irreguläre LDPC–Codes gehören zu den besten Kanalcodes überhaupt. Das gelbe Kreuz in der Grafik liegt praktisch auf der informationstheoretischen Grenzkurve für binäre Eingangssignale (grüne Kurve, mit BPSK beschriftet). Die Codewortlänge dieses irregulären Rate–1/2–Codes von **[CFRU01]** beträgt $2 \cdot 10^6$. Daraus ist schon ersichtlich, dass dieser Code nicht für den praktischen Einsatz gedacht war, sondern für einen Rekordversuch getunt wurde.

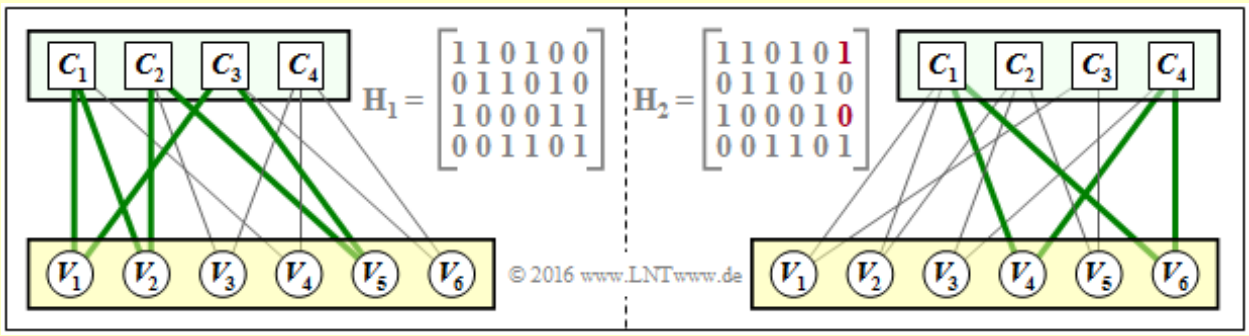


Bei der LDPC–Codekonstruktion geht man ja stets von der Prüfmatrix **H** aus. Für den gerade genannten Code hat diese die Dimension 1000000×2000000, beinhaltet also $2 \cdot 10^{12}$ Matrixelemente.

Füllt man die Matrix per Zufallsgenerator mit (wenigen) Einsen ⇒ „*Low–density*". So spricht man von *unstrukturiertem Code–Design*. Dies kann bei langen Codes zu folgenden Problemen führen:

- Die Komplexität des Coders kann zunehmen, da trotz Modifikation der Prüfmatrix **H** sichergestellt werden muss, dass die Generatormatrix **G** systematisch sein muss.
- Aufwändige Hardware–Realisierung des iterativen Decoders.
- „Error Floor" durch einzelne Einsen in einer Spalte (oder Zeile) sowie kurze Schleifen.

Beispiel: Im linken Teil der Grafik ist der Tanner–Graph für einen regulären LDPC–Code mit der Prüfmatrix $\mathbf{H}_1$ dargestellt. Grün eingezeichnet ist ein Beispiel für die minimale Schleifenlänge (englisch: *Girth*). Diese Kenngröße gibt an, wieviele Kanten man mindestens durchläuft, bis man von einem *Check Node* C_j wieder bei diesem landet (oder von V_i zu V_i). Im linken Beispiel ergibt sich die minimale Kantenlänge 6, zum Beispiel der Weg $C_1 \rightarrow V_1 \rightarrow C_3 \rightarrow V_5 \rightarrow C_2 \rightarrow V_2 \rightarrow C_1$.

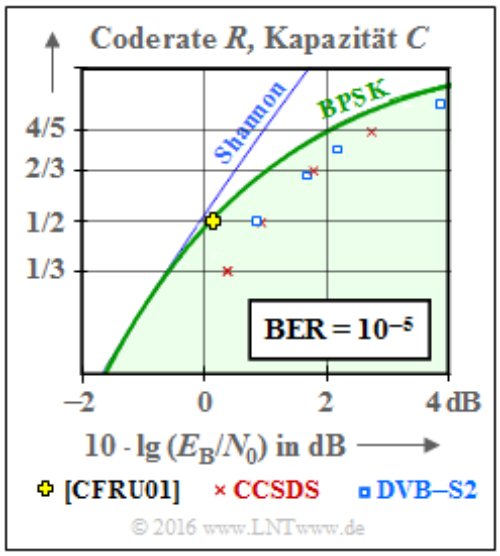


Vertauscht man in der Prüfmatrix nur zwei Einsen ⇒ Matrix $\mathbf{H}_2$, so ist die minimale Schleifenlänge 4, von $C_1 \rightarrow V_4 \rightarrow C_4 \rightarrow V_6 \rightarrow C_1$. Ein kleiner *Girth* führt zu einem „Error Floor" im BER–Verlauf.

Einige Anwendungsgebiete für LDPC–Codes

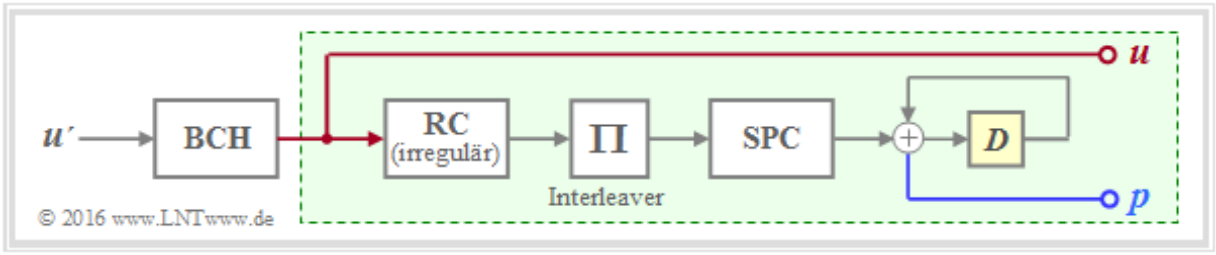
In dem Schaubild sind zwei Kommunikations–Standards, die auf strukturierten LDPC–Codes basieren, im Vergleich zur AWGN–Kanalkapazität eingetragen.

Anzumerken ist, dass für die eingezeichneten standardisierten Codes die Bitfehlerrate 10^{-5} zugrunde liegt, während die Kapazitätskurven (entsprechend der Informationstheorie) für die Fehlerwahrscheinlichkeit 0 gelten.



- Rote Kreuze zeigen die LDPC–Codes nach CCSDS (*Consultative Comittee for Space Data Systems*), entwickelt für ferne Weltraummissionen. Diese Klasse stellt Codes der Rate 1/3, 1/2, 2/3 und 4/5 bereit. Alle Punkte liegen ca. 1 dB rechts von der Kapazitätskurve für binären Eingang (grüne Kurve „BPSK").

- Die blauen Rechtecke markieren die LDPC–Codes für DVB–T2/S2. Die Abkürzungen stehen für „Digital Video Broadcasting – Satellite" bzw. „Digital Video Broadcasting – Terrestrial", und die „2" macht deutlich, dass es sich jeweils um die zweite Generation (von 2005 bzw. 2009) handelt. Der Standard ist durch 22 Prüfmatrizen definiert, die Raten von etwa 0.2 bis zu 19/20 zur Verfügung stellen. Je elf Varianten gelten für die Codelänge 64800 Bit (*Normal FECFRAME*) bzw. 16200 Bit (*Short FECFRAME*). Kombiniert mit **Modulationsverfahren hoher Ordnung** (8PSK, 16–ASK/PSK, ...) zeichnen sich die Codes durch große spektrale Effizienz aus.



DVB–Codes gehören zu den *Irregular Repeat Accumulate* (IRA) Codes die erstmals im Jahr 2000 in **[JKE00]** vorgestellt wurden. Die Grafik zeigt die Grundstruktur des Coders. Der grün hinterlegte Teil – mit Repetition Code (RC), Interleaver, Single Parity–Code (SPC) sowie Akkumulator – entspricht exakt einem seriell–verketteten Turbocode ⇒ siehe **RA–Coder.** Die Beschreibung des IRA–Codes basiert aber allein auf der Prüfmatrix **H**, die sich durch den *irregulären Repetition Code* in eine für die Decodierung günstige Form bringen lässt. Als äußerer Code wird zudem ein hochratiger BCH–Code (von *Bose–Chaudhuri–Hocquenghem*) verwendet, der den *Error Floor* herabsetzen soll.

In der oberen Grafik nicht eingetragen sind die LDPC–Codes für den Standard *DVB Return Channel Terrestrial* (RCS), für den WiMax–Standard (IEEE 802.16) sowie für das 10GBASE–T–Ethernet, die gewisse Ähnlichkeiten mit den IRA–Codes aufweisen.