# 9 Turboähnliche Kodes

## 9.1 LDPC[Low-Density Parity-Check]-(Block)Kodes

- 1962 von GALLAGER erstmals vorgestellt, trotz ihrer sehr guten Eigenschaften lange vergessen
  (Kodeverkettung schien überlegen, begrenzte Rechenleistung; FORNEY: „The way clearly far too complicated for the technology of the time.“ IEEE Spectrum, 2004.)
  1993 *Turbo Code*, zeigt Stärke iterativer Dekodierung auf
  1995 von MacKay/Neal leicht verbessert wiedererfunden und weiterentwickelt
- Grundlage für iterative Dekodierungsalgorithmen:
  Sparsame Kontrollmatrix $H_{k\times n}$ mit $w(H) \ll k \cdot n$
  (Für einen „guten“ Kode sollte aber Generatormatrix nicht sparsam sein: $w(a^*) = 1 \rightarrow w(a) \gg 1$)
- bei Vermeidung kleiner Zyklen in $H_{k\times n}$ an Leistung von ML Dekodierung
- Unterscheidung von LDPC-Kodes
  - regulär: Spaltengewicht $\gamma$ , Zeilengewicht $\rho$ konstant, $\gamma < \rho$ ;
    $$R = 1 - \frac{\gamma}{\rho} = \frac{l}{n}$$
    4-zyklenfrei: $d_{min} \geq \gamma + 1$
  - irregulär: Spalten- $\gamma_j$ , Zeilengewicht $\rho_i$ unterschiedlich;
    $$R = 1 - \frac{\frac{1}{n}\sum_{j=1}^{n}\gamma_j}{\frac{1}{k}\sum_{i=1}^{k}\rho_i} = \frac{l}{n}$$

## Beispiel

LDPC-Kode mit $\gamma = 3$, $\rho = 6$, $R = 1 - \frac{3}{6} = \frac{1}{2} = \frac{l}{n}$

$$H_{k\times n=6\times 12} = \begin{pmatrix} 1\,1\,0\,1\,0\,0\,0\,0\,1\,1\,0\,1 \\ 1\,1\,1\,1\,1\,0\,0\,1\,0\,0\,0\,0 \\ 1\,0\,1\,0\,1\,1\,0\,0\,0\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1\,1\,0\,1\,0\,1\,0 \\ 0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,0\,1 \end{pmatrix} = [H_l\, H_k]\,, \quad d_{min} = 4$$

In Vorbereitung der Anwendung iterativer Dekodierung:

- Zu $H$ definierte Indexmengen:

  $K_i = \{j : 1 \le j \le n, h_{ij} = 1\} \quad (i = 1, 2, ..., k)$, $|K_i| = \rho_{[i]}$
  Menge aller Variablen (Bits), die Einfluss auf die $i$-te Prüfgleichung haben

  Beispiel: $K_1 = \{1, 2, 4, 9, 10, 12\}$, $K_2 = \{1, 2, 3, 4, 5, 8\}$, ...

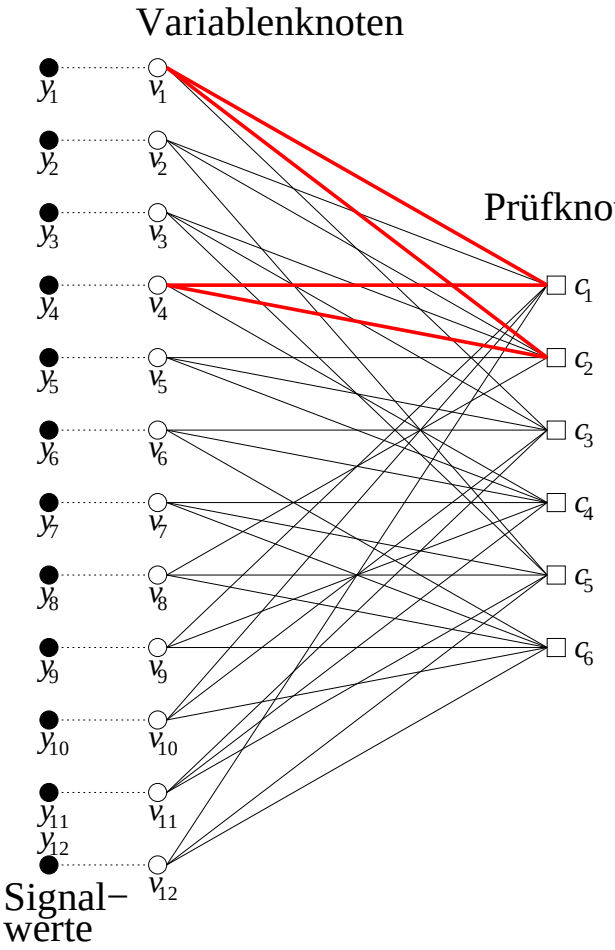
  $N_j = \{i : 1 \le i \le k, h_{ij} = 1\} \quad (j = 1, 2, ..., n)$, $|N_j| = \gamma_{[j]}$
  Menge der Prüfgleichungen, in welche die $j$-te Variable Einfluss hat
  Beispiel: $N_1 = \{1, 2, 3\}$, $N_2 = \{1, 2, 5\}$, $N_3 = \{2, 3, 5\}$, ...

  Anstelle $s = H \cdot b_h^T$ weniger Operationen mit

  $$s_i = \sum_{j\in K_i} y_{h,j} \bmod 2 \quad (i = 1, 2, ..., k),\ y_{h,j} \in \{0, 1\}$$

  (Einsparung von Multiplikationen, auch Additionen).

- Zu $H$ korrespondierender TANNER-Graph [factor, bipartite graph]




Jede(s) Variable (Bit) $v_j$ ($u_j$) ist in drei Prüfgleichungen enthalten, korrespondierend zu den drei Einsen in jeder Spalte.
Jede Prüfgleichung $c_i$ basiert auf sechs Variablen, korrespondierend zu den sechs Einsen in jeder Zeile.

Optimal: Graph ohne Zyklen [girth] $\longrightarrow$ Leistung einer ML Dekodierung

Ist das Gewicht der stellenweisen Multiplikation von zwei beliebigen Spalten (Zeilen) höchstens Eins, dann ist $H$ 4-zyklenfrei.

- $H$-Konstruktionen
  - zufallsähnlich (Vorgabe von Zyklenlänge, Gewichtsverteilung von Variablen-, Prüfknoten,..., z. B. mit PEG Verfahren)
  - strukturiert (algebraische Zshg.e, wie quasi-zyklische Kodes)

**Kodierung** (abhängig von $H$-Konstruktion)

- $a = a^* \cdot G = a_l \cdot G \quad (H \to G\,?)$
- $H_{k\times n} = [H_l\, H_k]$ und $a = [a_l\, a_k]$;
  Wegen $H \cdot a^T = \mathbf{0}$ ist $H_k \cdot a_k^T = H_l \cdot a_l^T$ und damit
  $\to a_k^T = \mathbf{H_k^{-1}} \cdot H_l \cdot a_l^T$
  (Voraussetzung: $H_k$ quadratisch und invertierbar)
- $H$ zyklisch:
  $a_j = \sum_{j' \in K_i \setminus j} a_{j'} \bmod 2 \quad (j = l + i,\ i = 1, 2, ..., k)$
- $H = [H_l\, H_k] = [H_1\, H_2]$
  $H_1$ zufallsähnlich; $H_2$ bidiagonal (staircase) (DVB-S2) oder
  $H_2$ untere Dreiecksmatrix (triangle):
  Kodierung wie $H$ zyklisch

**Dekodierung**

Dekodierungsmethode:

Iterative (Syndrom-)Dekodierung $\quad s = H \cdot b_h^T = \mathbf{0}\,?$

Dekodierungsalgorithmen:

- hard-decision Dekodierung:
  Basis: $y_{h,j} \in \{0, 1\}$
- hard/soft reliability-based (hybride) Dekodierung:
  Basis: $y_{h,j} \in \{0, 1\}$ und bei soft-Zuverlässigkeit Einbeziehung der Signalwerte $y_j \in \mathbb{R}$ in den Berechnungsablauf
- soft-decision Dekodierung:
  Basis: $y_j \in \mathbb{R}$

**hard-decision und soft/hard reliability-based Dekodierung**

Beispiel: $a_l = (110100)$ $a_k = (010011)$ $\longrightarrow$

$b_M =$(-0.9,-0.9, 1.0, 0.1, 0.3, 1.0, 0.8,-0.7, 1.0, 0.9,-1.0, 0.4), $b_h^{(0)} \in A$?

- **bit-flipping** Algorithmus (GALLAGER 1962)

  $s_i^{(0)} = \sum_{j \in K_i} y_{h,j}^{(0)} \bmod 2 \quad (i = 1, 2, ..., k)$

  $s^{(\tau)} \neq \mathbf{0}: \; e_j^{(\tau)} = \sum_{i \in N_j} s_i^{(\tau)} \quad (j = 1, 2, ..., n)$

  $\text{flip}(y_{h,j}^{(\tau+1)})$, wenn $e_j^{(\tau)} \geq T$ oder $= T_{max} = \max_{j=1}^{n} e_j^{(\tau)}$

  Abbruch: $s^{(\tau+1)} = \mathbf{0}$ oder $\tau > \tau_{max}$ , sonst $\tau = \tau + 1$

- hybride bit-flipping Algorithmen (2001 ... 2008 ...)

  – weigthed bit-flipping (WBF): bezieht Signalwerte $y_j$ als soft-Zuverlässigkeit einer Prüfgleichung ein:

  $w_i = \min_{j \in K_i} |y_j| \quad (i = 1, 2, ..., k)$ (QWBF: $w_i > \Delta : w_i = 2$, sonst $w_i = 1$)

  $s^{(\tau)} \neq \mathbf{0}: \; e_j^{(\tau)} = \sum_{i \in N_j} (2\, s_i^{(\tau)} - 1) \cdot w_i \quad (j = 1, 2, ..., n)$

  $\text{flip}(y_{h,j}^{(\tau+1)})$, wenn $e_j^{(\tau)} = T_{max}$

  – modified WBF (MWBF): bezieht Signalwerte als soft-Zuverlässigkeit einer Prüfgleichung und Bit-basiert ein ($\alpha$ – Wichtungsfaktor, exper. ermittelt, wächst mit $\gamma$: $\alpha \in [0.3, 2.0]$):

  $w_i = \min_{j \in K_i} |y_j| \quad (i = 1, 2, ..., k)$

  $s^{(\tau)} \neq \mathbf{0}: \; e_j^{(\tau)} = \sum_{i \in N_j} (2\, s_i^{(\tau)} - 1) \cdot w_i - \alpha\, |y_j| \quad (j = 1, 2, ..., n)$

  $\text{flip}(y_{h,j}^{(\tau+1)})$, wenn $e_j^{(\tau)} = T_{max}$

– improved MWBF (IMWBF): wie modified, bezieht aber extrinsische Information als soft-Zuverlässigkeit ein

$$L_e(\widehat{u}_{ij}) = w_{ij} = \min_{j' \in K_i \setminus j} |y_{j'}| \quad (i = 1, 2, ..., k)$$

$$s^{(\tau)} \neq \mathbf{0} : e_j^{(\tau)} = \sum_{i \in N_j} (2s_i^{(\tau)} - 1) w_{ij} - \alpha |y_j| \quad (j = 1, 2, ..., n)$$

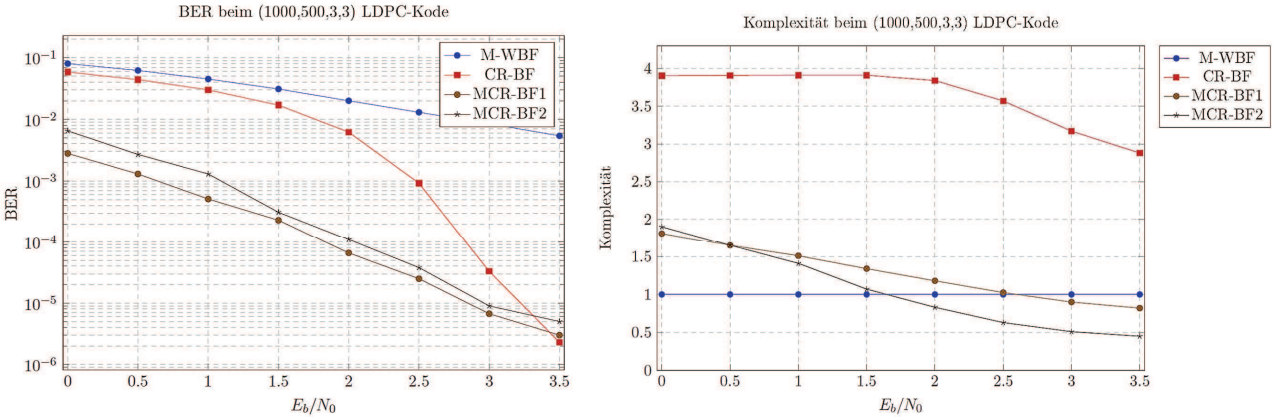
$$\text{flip}(y_{h,j}^{(\tau+1)}), \text{ wenn } e_j^{(\tau)} = T_{max}$$

– iterative Anpassung der Prüfgleichungs-Zuverlässigkeit[10] über alle $j_{\text{flip}}^{(\tau)}$ nicht erfüllte Prüfgleichungen $s_i^{(\tau)} = 1$:

$$w_i^{(\tau+1)} = w_i^{(\tau)} + \delta \quad \text{bzw.} \quad w_{ij}^{(\tau+1)} = w_{ij}^{(\tau)} + \delta \qquad (\delta \approx 0.03)$$

⋮

– C.Y. Chang, et. al. *Check Reliability Based Bit-Flipping Decoding Algorithms for LDPC Codes.* 2010, u.a. untersucht in:
A. Kropp *Einordnung und Bewertung jüngst modifizierter BF/MLG Dekodierungsalgorithmen.* Bachelorarbeit 2014




→ modifizierte Versionen (MCR-BF1,MCR-BF2) kippen in Abhängigkeit von $w(s)$ mehrere Bits

[10] D. Qian, et al. *A Modification to Weighted Bit-Flipping Decoding Algorithm for LDPC Codes based on Reliability Adjustment.* 2008 IEEE

Kontrollmatrix **orthogonal**[11]: $\gamma = \rho$ in $\mathbf{H_{n \times n}}$, $d_{min} = \gamma + 1$
$\rightarrow$ **majority-logic** (MLG) Algorithmus

Beispiel: (15,7,5)BCH, orthogonal

$g(x) = x^8 + x^7 + x^6 + x^4 + 1 \rightarrow h(x) = \frac{f(x)}{g(x)} = x^7 + x^6 + x^4 + 1$
$a = (100010111000000) \quad \longrightarrow \quad b_h = (111010111000000) \in A?$

- one-step majority-logic Algorithmus (REED 1954, MASSEY 1963)
  Notw.: orthogonale Prüfsumme bezogen auf Variable (Bit) $v_j$

  $$s_i = \sum_{j \in K_i} y_{h,j} \bmod 2 \quad (i = 1, 2, ..., k)$$

  $$s \neq \mathbf{0}: \; e_j = \sum_{i \in N_j} s_i \rightarrow \text{flip}(y_{h,j}), \text{ wenn } e_j > \lfloor \tfrac{\gamma}{2} \rfloor \qquad (j = 1, 2, ..., n)$$

  Auch möglich:

  $$s \neq \mathbf{0}: \; e_j = \sum_{i \in N_j} (2\, s_i - 1) \text{ mit } e_j \in [-\gamma, +\gamma]$$
  $$\rightarrow \text{flip}(y_{h,j}), \text{ wenn } e_j > 0 \qquad (j = 1, 2, ..., n)$$

  $\rightarrow$ Überprüfung von $s = \mathbf{0}$? zum Erkennen von Dekodierungsversagen!

[11] Summiere $v_j$ beeinflussende Prüfgleichungen: Summe über $v_j$ ist $\gamma$, sonst höchstens 1.

- hard **r**eliability-based **iterativer** MLG Algorithmus (2009 [12])

  $r_j^{(0)} = \gamma, \;\; \text{wenn} \;\; y_{h,j}^{(0)} = 0, \;\; \text{sonst} \;\; r_j^{(0)} = -\gamma$

  $s_i^{(0)} = \sum_{j \in K_i} y_{h,j}^{(0)} \bmod 2 \quad (i = 1, 2, ..., k)$

  $s^{(\tau)} \neq \mathbf{0} : \left[ L_e(\widehat{u}_{ij}) = \sum_{j' \in K_i \setminus j} y_{h,j'}^{(\tau)} \bmod 2 = s_i^{(\tau)} \oplus y_{h,j}^{(\tau)} \;\; (i = 1..k) \right]$

  $e_j^{(\tau)} = \sum_{i \in N_j} (2\, L_e(\widehat{u}_{ij}) - 1) = \sum_{i \in N_j} (2(s_i^{(\tau)} \oplus y_{h,j}^{(\tau)}) - 1)$

  $r_j^{(\tau+1)} = \min\{\max\{r_j^{(\tau)} - e_j^{(\tau)}, -\gamma\}, \gamma\}$

  $y_{h,j}^{(\tau+1)} = 0, \;\; \text{wenn} \;\; r_j^{(\tau+1)} \geq 0, \;\; \text{sonst} \;\; y_{h,j}^{(\tau+1)} = 1 \qquad (j = 1..n)$

  Abbruch: $s^{(\tau+1)} = \mathbf{0}$ oder $\tau > \tau_{max}$ , sonst $\tau = \tau + 1$

- soft reliability-based iterativer MLG Algorithmus

  soft aus $(2^x - 1)$ quantisiertem Signalbereich ($x\,Bit$ quantis.):

  $r_j^{(0)} = y_{q,j} \in [-(2^{x-1}-1), 2^{x-1}-1], y_{q,j} = \text{round}\,(y_j \cdot (2^{x-1}-1)) \quad (j = 1..n)$

  sonst wie hard

**Vergleich** WBF- und iterative MLG Algorithmen

– MLG erfordern nur logische Operationen und Integer-Additionen
– MLG erlauben parallele Dekodierung
– MLG nur für orthogonale $H$
– hard MLG vergleichbar mit BF, schlechter als WBF
– soft MLG konvergiert wesentlich schneller als WBF
– soft MLG nur $< 1$ dB zum optimalen sum-product Algorithmus

[12] Q. Huang, et al. *Two Reliability-Based Iterative Majority-Logic Decoding Algorithms for LDPC Codes.* 2009 IEEE