Irregular Repeat Accumulate- and Low-Density Parity-Check-Codes

Janis Dingel

6.4.2005 / JASS05 - St. Petersburg
1 Introduction
1 Introduction
2 Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code

3 Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code

4 RA vs. LDPC
   - Performance
   - Complexity

5 Punctured RA-Codes

6 Conclusion
1 Introduction
2 Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3 Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
Content

1. Introduction
2. Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3. Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4. RA vs. LDPC
   - Performance
   - Complexity
1 Introduction
2 Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3 Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4 RA vs. LDPC
   - Performance
   - Complexity
5 Punctured RA-Codes
Outline

Content

1 Introduction
2 Presentation of the Channel Codes
   • Low-Density Parity-Check Code
   • Repeat-Accumulate-Code
3 Decoding
   • Decoding using the Sum Product Algorithm
   • Decoding the LDPC-Code
   • Decoding the RA-Code
4 RA vs. LDPC
   • Performance
   • Complexity
5 Punctured RA-Codes
6 Conclusion
LDPC-Codes invented 1960! by Gallager...
LDPC-Codes invented 1960! by Gallager...
..then largely forgotten for decades!
Introduction I

- LDPC-Codes invented 1960! by Gallager...
- ..then largely forgotten for decades!

🔥 1993: Invention of Turbo-Codes 🔥
⇒ rediscovery of LDPC Codes by David J.C. MacKay 1999
Introduction

LDPC-Codes invented 1960! by Gallager...
..then largely forgotten for decades!

1993: Invention of Turbo-Codes
⇒ rediscovery of LDPC Codes by David J.C. MacKay 1999
2001: LDPC Codes within 0.00045dB of the Shannon limit
1998: Divsalar et al. study a very simple class of "Turbo-like" Codes for theoretical purpose
1998: Divsalar et al. study a very simple class of "Turbo-like" Codes for theoretical purpose
While very simple, the codes show a very good performance
1998: Divsalar et al. study a very simple class of "Turbo-like" Codes for theoretical purpose

While very simple, the codes show a very good performance

They call them "Repeat-Accumulate"-Codes
1998: Divsalar et al. study a very simple class of "Turbo-like" Codes for theoretical purpose

While very simple, the codes show a very good performance

They call them "Repeat-Accumulate"-Codes

Present: RA Codes are used in today's and future standards (DVB-S2, WirelessMAN IEEE802.16)
Properties of Repeat-Accumulate Codes:

- very simple encoding procedure
- complexity linear with blocklength
  ⇒ very interesting for practical applications
Properties of Repeat-Accumulate Codes:

- very simple encoding procedure
- complexity linear with blocklength
  \( \Rightarrow \) very interesting for practical applications

- RA-Codes can be as good as the best known LDPC-Codes
  (has been shown).
Introduction III

Properties of Repeat-Accumulate Codes:

- very simple encoding procedure
- complexity linear with blocklength
  ⇒ very interesting for practical applications
- RA-Codes can be as good as the best known LDPC-Codes (has been shown).
- RA-Codes are not an independent class of codes.
- RA-Codes are a subclass of LDPC- and of Turbo-Codes.
  ⇒ two different decoding methods can be applied (which one is better?)
Outline

1 Introduction

2 Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code

3 Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code

4 RA vs. LDPC
   - Performance
   - Complexity

5 Punctured RA-Codes

6 Conclusion
LDPC Codes are Blockcodes!

Blockcodes:
- Defined by the Parity-Check Matrix $\mathbf{H} \in \{0, 1\}^{(N-K) \times N}$
- The Code $\mathcal{C}$ consists of all $\mathbf{x}$ that satisfy $\mathbf{H}\mathbf{x} = \mathbf{0}$.

Example: $(7,4,3)$ Hamming Code

$$
\mathbf{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
\end{pmatrix}
\iff
\begin{cases}
x_1 \oplus x_2 \oplus x_3 \oplus x_5 = 0 \\
x_2 \oplus x_3 \oplus x_4 \oplus x_6 = 0 \\
x_3 \oplus x_4 \oplus x_5 \oplus x_7 = 0
\end{cases}
$$
A Generatormatrix maps an information vector $u$ to a Codeword

$$x = G^T u.$$  

$$\Rightarrow 0 = HG^T u$$

$$\Rightarrow 0 = HG^T$$

We only consider systematic codewords: $x = [u|p]^T$

$$G = [I_{K\times K}|P_{K\times(N-K)}]$$

$$\Rightarrow HG^T = H_1 I + H_2 P^T = 0$$

$$\Rightarrow P^T = H_2^{-1} H_1.$$
Encoding II

Encoding with given Parity-Check Matrix:

\[ x = G^T u = \begin{bmatrix} I \\ P^T \end{bmatrix} u = \begin{bmatrix} u \\ H_2^{-1}H_1u \end{bmatrix}. \]

Figure: Encoding procedure for LDPC (Block) Codes.
From Blockcodes to LDPC-Codes

Extensions to LDPC-Codes:
From Blockcodes to LDPC-Codes

Extensions to LDPC-Codes:

- The PC-Matrix $H$ is very big!
- The PC-Matrix is sparse (low density).
From Blockcodes to LDPC-Codes

Extensions to LDPC-Codes:
- The PC-Matrix $H$ is very big!
- The PC-Matrix is sparse (low density).
- Example: $N=20000$, $R=1/2$, column weight 3, row weight 6
Extensions to LDPC-Codes:

- The PC-Matrix $H$ is very big!
- The PC-Matrix is sparse (low density).
- Example: $N=20000$, $R=1/2$, column weight 3, row weight 6

High encoding complexity arises from the high density in $H_{2-1}$.
Graphical representation I

\[
\begin{bmatrix}
1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
\end{bmatrix}
\]

Figure: Graph of Hamming Code.
Graphical representation II

Figure: Graph of LDPC Codes.

Notation

- $dv_i$: degree of $i$th variable node
- $dc_i$: degree of $i$th check node
- $\Pi$: edge interleaver

Regular code:

- $dv_i = c_1 \forall i$
- $dc_i = c_2 \forall i$
Outline

1. Introduction
2. Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3. Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4. RA vs. LDPC
   - Performance
   - Complexity
5. Punctured RA-Codes
6. Conclusion
RA Code is a special LDPC Code

Practical interest due to low encoding complexity:

\[ x = G^T u = \begin{bmatrix} I \\ P^T \end{bmatrix} u = \begin{bmatrix} u \\ H_2^{-1} H_1 u \end{bmatrix}. \]

RA Code special structure:

\[ H_2 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \Rightarrow H_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \]
RA-Encoding

Figure: Block diagram of the simplified encoding.

Example Code: Intel proposal for IEEE802.16 rate 4/5 $N = 2000$
Introduction
Presentation of the Channel Codes
Decoding
RA vs. LDPC
Punctured RA-Codes
Conclusion

Graphical representation of RA Codes

Figure: Graph of RA Codes.
Outline

1 Introduction
2 Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3 Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4 RA vs. LDPC
   - Performance
   - Complexity
5 Punctured RA-Codes
6 Conclusion
Decoding using the Sum Product Algorithm
Decoding the LDPC-Code
Decoding the RA-Code

Information from the Channel
SPSA for Block Codes

\[ \Pr(x_3 = 0 \mid y_3) = \Pr(x_1 = 0 \mid y_1) \Pr(x_2 = 0 \mid y_2) \]

\[ L_{x_3} = L_{x_2} + L_{x_3} \]

\[ \Pr(x_3 = 0 \mid y_3) = \Pr(x_1 = 0 \mid y_1) \Pr(x_2 = 0 \mid y_2) + \Pr(x_1 = 1 \mid y_1) \Pr(x_2 = 1 \mid y_2) \]

\[ L_{x_3} = L_{x_1} \oplus L_{x_2} \]

\[ = 2 \tanh^{-1}(\tanh(\frac{L_{x_1}}{2}) \tanh(\frac{L_{x_2}}{2})) \]
SPA for LDPC Codes

The SPA delivers exact a posteriori probabilities if the graph is cycle free.
SPA for LDPC Codes

- The SPA delivers exact a posteriori probabilities if the graph is cycle free.
- LDPC graphs are not cycle free ⇒ infinite message passing.
- Even in a fixpoint: no accurate probabilities
The SPA delivers exact a posteriori probabilities if the graph is cycle free.

LDPC graphs are not cycle free ⇒ infinite message passing.

Even in a fixpoint: no accurate probabilities

We don’t care. We just want the right codeword

⇒ ”Apply the rules and hope for the best”
SPA for LDPC Codes

- The SPA delivers exact a posteriori probabilities if the graph is cycle free.
- LDPC graphs are not cycle free ⇒ infinite message passing.
- Even in a fixpoint: no accurate probabilities
- We don’t care. We just want the right codeword
⇒ ”Apply the rules and hope for the best”
- In practice: It works!
  - timing schedule needed
  - restrictions to code design apply.
Outline

1. Introduction
2. Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3. Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4. RA vs. LDPC
   - Performance
   - Complexity
5. Punctured RA-Codes
6. Conclusion
Update Rule Variable Nodes

\[ L_{intr_j} = L_c y_j + \sum_{i=1}^{dv_j} L_{apr_i} \quad \forall j = 1, \ldots, N \]

\[ L_{extr_i} = L_{intr_j} - L_{apr_i} \quad \forall i = 1, \ldots, dv_i, j = 1, \ldots, N \]
Update Rule Check Nodes

\[ L_{extr_k} = \sum_{i=1}^{dc_j} \bigoplus L_{apr_i} \quad \forall k. \]
Turbo-Style view

Variable node decoder (repetitions) → Interleaving → Check node decoder (parity checks) → Hard decision → Sink

From AWGN channel
Outline

1. Introduction
2. Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3. Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4. RA vs. LDPC
   - Performance
   - Complexity
5. Punctured RA-Codes
6. Conclusion
RA = LDPC + Accumulator
Efficient Accumulator decoding

(a). Message flow

(b). Forward pass

(c). Backward pass

Compute...

1. forward messages
   \[ L_{f_i} = (L_c y_{i-1} + L_{f_{i-1}}) \boxplus L_o(p'_i) \]

2. backward messages
   \[ L_{b_i} = (L_c y_{i+1} + L_{b_{i+1}}) \boxplus L_o(p'_{i+1}) \]

3. outgoing messages
   \[ L_e(p'_i) = (L_{f_{i-1}} + L_c y_{i-1}) \boxplus (L_{b_i} + L_c y_i) \]
Turbo view

accumulator and check node decoder

APP decoding (memory one trellis) check node decoder (parity check codes)

from AGWN channel systematic part (length k)

MUX

ACC

CND

edge interleaving

variable node decoder (repetition codes)

sink

hard decision

TUM

Janis Dingel
Irregular RA-Codes and LDPC-Codes
Outline

1. Introduction
2. Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3. Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4. RA vs. LDPC
   - Performance
   - Complexity
5. Punctured RA-Codes
6. Conclusion
IEEE802.16 proposals

Intel R4/5, N=2000

Nortel R8/9, N=2634

LG R1/2, N=2000

DVB-S2 Standards

Motivation

Nowadays applications demand low encoding complexity.

⇒ Many proposed PC-Matrices have lower staircase structure

⇒ Can be seen as both, LDPC and RA-Code.

⇒ 2 different decoding schemes can be applied.

⇒ Question: Which one is better?
Intel proposal, Rate 1/2, N=800, BER
Intel proposal, Rate 1/2, N=800, FER
Intel proposal, Rate 4/5, N=2000, BER
Intel proposal, Rate 4/5, N=2000, FER

![Graph showing FER performance of Intel proposal for different rates and iterations.](image)
LG proposal, Rate 1/2, N=800, BER

Rate 0.5

BER

10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}

E_b/N_0 in dB

Uncoded transmission
LG; N=800; 10 Iterations
LG; N=800; 30 Iterations
LG; N=800; 100 Iterations
LG RA; N=800; 10 Iterations
LG RA; N=800; 30 Iterations
LG RA; N=800; 100 Iterations
LG proposal, Rate 1/2, $N=800$, FER

![Graph showing FER vs. Eb/N0 in dB for different iterations and code rates (LG and LG RA)]
DVB-S2, $N=64800$, 20 Iterations

![Graph showing BER vs. Eb/N0 for different transmission schemes.]

- Uncoded transmission
- DVBS2; Rate 1/2
- DVBS2 - RA; Rate 1/2
- DVBS2; Rate 2/3
- DVBS2 - RA; Rate 2/3
- DVBS2; Rate 3/4
- DVB-S2 - RA; Rate 3/4
Outline

1 Introduction
2 Presentation of the Channel Codes
   • Low-Density Parity-Check Code
   • Repeat-Accumulate-Code
3 Decoding
   • Decoding using the Sum Product Algorithm
   • Decoding the LDPC-Code
   • Decoding the RA-Code
4 RA vs. LDPC
   • Performance
   • Complexity
5 Punctured RA-Codes
6 Conclusion
RA decoding better performance. Just a question of effort?

Let’s count the numbers of operations for one iteration.

**# of Additions for LDPC decoding:**

For every variable node: 
\[ L_{intr_j} = L_{cy_j} + \sum_{i=1}^{dv_j} L_{apr_i} \]

Extrinsic Information: 
\[ L_{extr_j} = L_{ch_i} + \sum_{k=1 \atop k \neq j}^{dv_i} L_{apriori_k} \]

\[
\#ADD_{LDPC} = 2 \sum_{i=1}^{N} dv_i = 2 NNZ(H) := 2c
\]
Figure: Efficient computation of checknode messages.

⇒ $3(d_{ci} - 2)$ Boxplus operations per checknode

# of Boxplus operations for LDPC decoding

\[
\#BXP_{LDPC} = 3 \sum_{i=1}^{N-K} (d_{ci} - 2) = 3c - 6(N - K)
\]
# of Additions in LDPC part for RA decoding:

Only K variable nodes!

\[ \#ADD_{RA1} = 2 \sum_{i=1}^{K} dv_i \]
# of Boxplus operations in LDPC part for RA decoding:

\[ \# BX P_{RA1} = 3 \sum_{i=1}^{N-K} (\tilde{d}_c - 1) \]
Remember special structure of $H$:

\[
\tilde{d}c_1 = dc_1 - 1, \\
\tilde{d}c_i = dc_i - 2 \quad \forall i = 2, \ldots, N - K
\]

\[
\Rightarrow \sum_{i=1}^{N-K} \left(3\tilde{d}c_i - 3\right) = 3 \sum_{i=1}^{N-K} \tilde{d}c_i - 3(N - K)
\]

\[
= 3 dc_1 - 3 + 3 \sum_{i=2}^{N-K} dc_i - 6(N - K - 1) - 3(N - K)
\]

\[
= 3 \sum_{i=1}^{N-K} dc_i - 9(N - K) + 3
\]

\[
\#BXP_{RA1} = 3c - 9(N - K) + 3
\]
### Introduction

Presentation of the Channel Codes

Decoding

RA vs. LDPC

Punctured RA-Codes

Conclusion

### Performance

### Complexity

<table>
<thead>
<tr>
<th>Computation</th>
<th>#Additions</th>
<th>#Bxpoperations</th>
<th>#total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>N-K-2</td>
<td>N-K-2</td>
<td>2(N-K)-4</td>
</tr>
<tr>
<td>Backward</td>
<td>N-K-2</td>
<td>N-K-1</td>
<td>2(N-K)-3</td>
</tr>
<tr>
<td>Outgoing</td>
<td>2</td>
<td>(N-K-1)</td>
<td>N-K+1</td>
</tr>
<tr>
<td>total</td>
<td>2(N-K)-2</td>
<td>3(N-K)-4</td>
<td>5(N-K)-6</td>
</tr>
</tbody>
</table>

Table: Number of operations needed for accumulator
Number of Operations of RA decoding

\[
\#ADD_{RA} = 2 \sum_{i=1}^{K} dv_i + 2(N - K) - 2
\]
\[
\#BXP_{RA} = 3c - 6(N - K) - 1
\]
Number of Operations of RA decoding

$\#ADD_{RA} = 2 \sum_{i=1}^{K} dv_i + 2(N - K) - 2$

$\#BXP_{RA} = 3c - 6(N - K) - 1$

Number of Operations of LDPC decoding

$\#ADD_{LDPC} = 2c$

$\#BXP_{LDPC} = 3c - 6(N - K)$
Number of RA BXP operations

\[ \#BXP_{RA} = 3c - 6(N - K) - 1 \]

Number of LDPC BXP operations

\[ \#BXP_{LDPC} = 3c - 6(N - K) \]

RA wins!
Number of RA Additions

\[ \#ADD_{RA} = 2 \sum_{i=1}^{K} dv_i + 2(N - K) - 2 \]

Number of LDPC Additions

\[ \#ADD_{LDPC} = 2c \]
Number of RA Additions

\[ \#ADD_{RA} = 2 \sum_{i=1}^{K} dv_i + 2(N - K) - 2 \]

Number of LDPC Additions

\[ \#ADD_{LDPC} = 2c \]

Remember special structure:

\[ c = \sum_{i=1}^{K} dv_i + 2(N - K - 1) + 1 \]
**Number of RA Additions**

\[
\#ADD_{RA} = 2 \sum_{i=1}^{K} dv_i + 2(N - K) - 2
\]

**Number of LDPC Additions**

\[
\#ADD_{LDPC} = 2c = 2 \sum_{i=1}^{K} dv_i + 2(N - K) - 2
\]

Remember special structure:

\[
c = \sum_{i=1}^{K} dv_i + 2(N - K - 1) + 1
\]
**Number of RA Additions**

\[ \#ADD_{RA} = 2 \sum_{i=1}^{K} dv_i + 2(N - K) - 2 \]

**Number of LDPC Additions**

\[ \#ADD_{LDPC} = 2c \]

\[ = 2 \sum_{i=1}^{K} dv_i + 4(N - K) - 2 \]

Remember special structure:

\[ c = \sum_{i=1}^{K} dv_i + 2(N - K - 1) + 1 \]

\[ \Rightarrow \frac{\#ADD_{RA}}{\#ADD_{LDPC}} < 1 \quad \forall R < 1 \]

"RA wins!"
Result of complexity analysis

Independent of the Rate and independent of the entries in the Parity-Check Matrix $H$: The RA decoder takes less operations.
Result of complexity analysis

Independent of the Rate and independent of the entries in the Parity-Check Matrix $H$: The RA decoder takes less operations.

Not a sufficient complexity analysis:

- Only no. of operations considered here.
- Hardware implementation.
- Parallelization issues.
”Puncturing shall be used for flexible rate adjustment”

LG proposal for IEEE802.16:

Figure 4. An example of reducing parity bits (shaded area is reduced)
leads to bad results
Maybe we can do better?

We can recover the punctured bits (don’t puncture the first one):

\[
Hx = [H_1 | H_2]x = H_1 u + H_2 p = Ip' + H_2 p = 0.
\]

\[
\begin{bmatrix}
  p'_1 \\
  p'_2 \\
  p'_3 \\
  \vdots \\
  p'_{N-K}
\end{bmatrix}
+ \begin{bmatrix}
  1 & 0 & 0 & 0 & \cdots & 0 \\
  1 & 1 & 0 & 0 & \cdots & 0 \\
  0 & 1 & 1 & 0 & \cdots & 0 \\
  0 & 0 & 1 & 1 & \cdots & 0 \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
  x_{K+1} \\
  x_{K+2} \\
  x_{K+3} \\
  x_{K+4} \\
  \vdots \\
  x_N
\end{bmatrix} = 0.
\]

Recovery (assume the last 100 bits have been punctured):

\[
x_{N-99} = x_{N-100} \oplus p'_{N-K-99}
\]

\[
x_{N-98} = x_{N-99} \oplus p'_{N-K-98}
\]

\[
\vdots
\]
At the decoder: punctured Bit sets all outgoing check node messages to zero.

⇒ First approach: preprocessing.

\[
L(p_i) = \left( \sum_{j=1}^{K} H_{i,j} L\text{cy}_j \right) \oplus L(p_{i-1})
\]
Decoding

At the decoder: punctured Bit sets all outgoing check node messages to zero.

⇒ First approach: preprocessing.

\[
L(p_i) = \left( \sum_{j=1}^{K} H_{i,j} L_{cy} \right) \oplus L(p_{i-1})
\]

**Problem:** Propagation of uncertainty. converges to zero rapidly.
**Reason:** Punctured Bit \( p_i \) depends on punctured Bit \( p_{i-1} \)
**Solution:** Find smarter puncturing scheme.
New puncturing scheme: puncture every second bit.

⇒ punctured Bit can be recovered with information from transmitted Bit.

⇒ Preprocessing + LDPC decoding:

\[
L(p_{2i}) = \left( \sum_{j=1}^{K} H_{i,j} L_{c} y_{j} \right) \boxplus L(p_{2i-1})
\]

\[
L_{c} y_{K+2i} = \left( \sum_{j=1}^{K} H_{i,j} L_{c} y_{j} \right) \boxplus L_{c} y_{K+2i-1}
\]
New puncturing scheme: puncture every second bit.

⇒ punctured Bit can be recovered with information from transmitted Bit.

⇒ Preprocessing + LDPC decoding:

\[
L(p_{2i}) = \left( \sum_{j=1}^{K} \bigoplus H_{i,j} L_{cy_j} \right) \bigoplus L(p_{2i-1})
\]

\[
L_{cy_{K+2i}} = \left( \sum_{j=1}^{K} \bigoplus H_{i,j} L_{cy_j} \right) \bigoplus L_{cy_{K+2i-1}}
\]

⇒ or ”doped” RA decoding (preprocessing done automatically).
Puncture every second Bit from 201 to 400 \(\Rightarrow\) 100 pBits
More encouraging results

![Graph showing BER vs. Eb/N0 for different code types and decoding methods.](Image)

- **Rate 0.88889**
- **Uncoded transmission**
- **LG whole block punctured – normal ldpc decoding; N=1800; 100 Iterationen**

Janis Dingel
Irregular RA-Codes and LDPC-Codes

TUM
More encouraging results

![Graph showing BER vs. Eb/N0 for different transmission methods: Uncoded transmission, LG whole block punctured with normal LDPC decoding, LG smart punctured with preprocessing, and LDPC with preprocessing. The graph is labeled Rate 0.88889.]

Janis Dingel

Irregular RA-Codes and LDPC-Codes
More encouraging results

Rate 0.88889

- Uncoded transmission
- LG whole block punctured – normal ldpc decoding; N=1800; 100 Iterationen
- LG smart punctured – ldpc with preprocessing; N=1800; 100 Iterationen
- LG smart punctured; doped RA decoding; N=1801; 100 Iterationen
RA Codes can be seen as both, LDPC and Turbo Codes.
The RA decoding method seems to have better performance (especially for low rates and large Blocklength).
RA decoding needs less operations per iteration.
Puncturing of RA codes can be done in 2 ways, smart and not so smart.