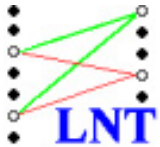
Irregular Repeat Accumulate- and Low-Density Parity-Check-Codes

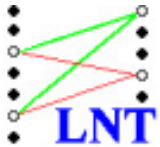
Janis Dingel

6.4.2005 / JASS05 - St. Petersburg



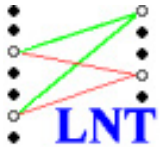
Content

1 Introduction



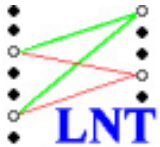
Content

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



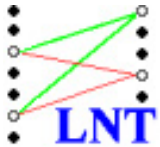
Content

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



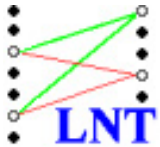
Content

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



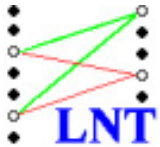
Content

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



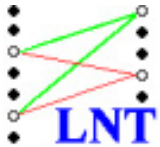
Content

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



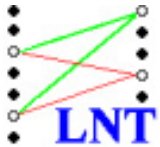
Introduction I

- LDPC-Codes invented 1960! by Gallager...



Introduction I

- 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 

$\Rightarrow$ rediscovery of LDPC Codes by David J.C. MacKay 1999

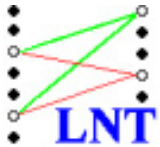
Introduction I

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

 1993: Invention of Turbo-Codes 

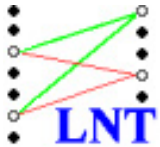
$\Rightarrow$ rediscovery of LDPC Codes by David J.C. MacKay 1999

- 2001: LDPC Codes within 0.00045dB of the Shannon limit



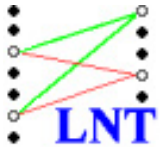
Introduction II

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



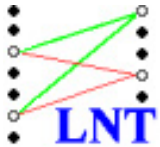
Introduction II

- 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



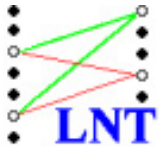
Introduction II

- 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



Introduction II

- 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)

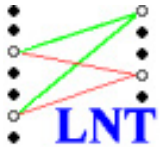


Introduction III

Properties of Repeat-Accumulate Codes:

- very simple encoding procedure
- complexity linear with blocklength

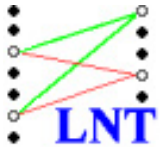
  $\Rightarrow$ very interesting for practical applications



Introduction III

Properties of Repeat-Accumulate Codes:

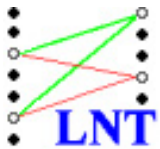
- very simple encoding procedure
- complexity linear with blocklength
  $\Rightarrow$ very interesting for practical applications
- RA-Codes can be as good as the best known LDPC-Codes (has been shown).



Introduction III

Properties of Repeat-Accumulate Codes:

- very simple encoding procedure
- complexity linear with blocklength
  - $\Rightarrow$ 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.
  - $\Rightarrow$ 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

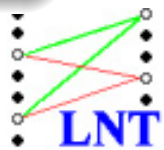
LDPC Codes are Blockcodes!

Blockcodes:

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

Example: (7,4,3) Hamming Code

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \Leftrightarrow \begin{cases} x_1 \oplus x_2 \oplus x_3 \oplus x_5 = 0 \\ x_2 \oplus x_3 \oplus x_4 \oplus x_6 = 0 \\ x_3 \oplus x_4 \oplus x_5 \oplus x_7 = 0 \end{cases}$$



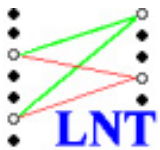
Encoding I

A Generatormatrix maps an information vector $\boldsymbol{u}$ to a Codeword

$$\boldsymbol{x} = \boldsymbol{G}^T \boldsymbol{u}.$$
$$\Rightarrow \mathbf{0} = \boldsymbol{H}\boldsymbol{G}^T \boldsymbol{u}$$
$$\Rightarrow \mathbf{0} = \boldsymbol{H}\boldsymbol{G}^T$$

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

$$\boldsymbol{G} = [\boldsymbol{I}_{K\times K}|\boldsymbol{P}_{K\times(N-K)}]$$
$$\Rightarrow \boldsymbol{H}\boldsymbol{G}^T = \boldsymbol{H_1}\boldsymbol{I} + \boldsymbol{H_2}\boldsymbol{P}^T = \mathbf{0}$$
$$\Rightarrow \boldsymbol{P}^T = \boldsymbol{H_2}^{-1}\boldsymbol{H_1}.$$



Encoding II

Encoding with given Parity-Check Matrix:

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

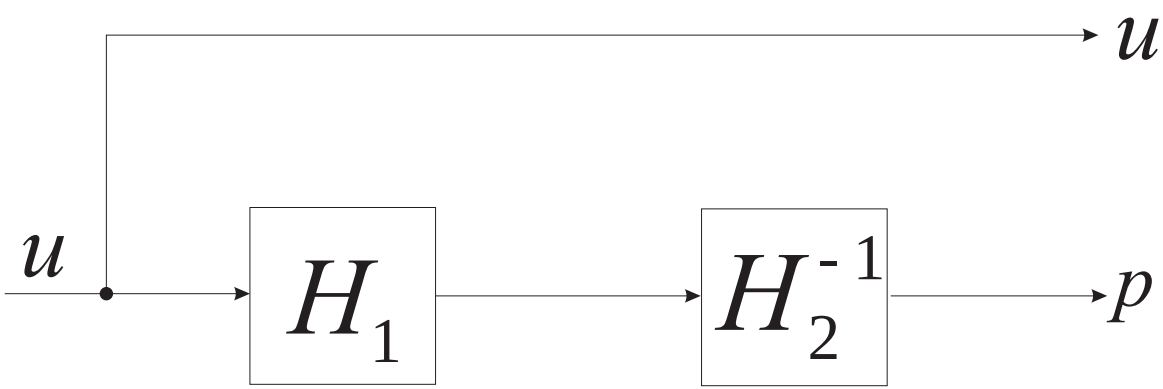
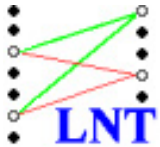


Figure: Encoding procedure for LDPC (Block) Codes.

From Blockcodes to LDPC-Codes

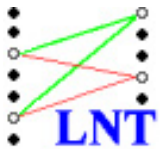
Extensions to LDPC-Codes:



From Blockcodes to LDPC-Codes

Extensions to LDPC-Codes:

- The PC-Matrix $\boldsymbol{H}$ is very big!
- The PC-Matrix is sparse (low density).

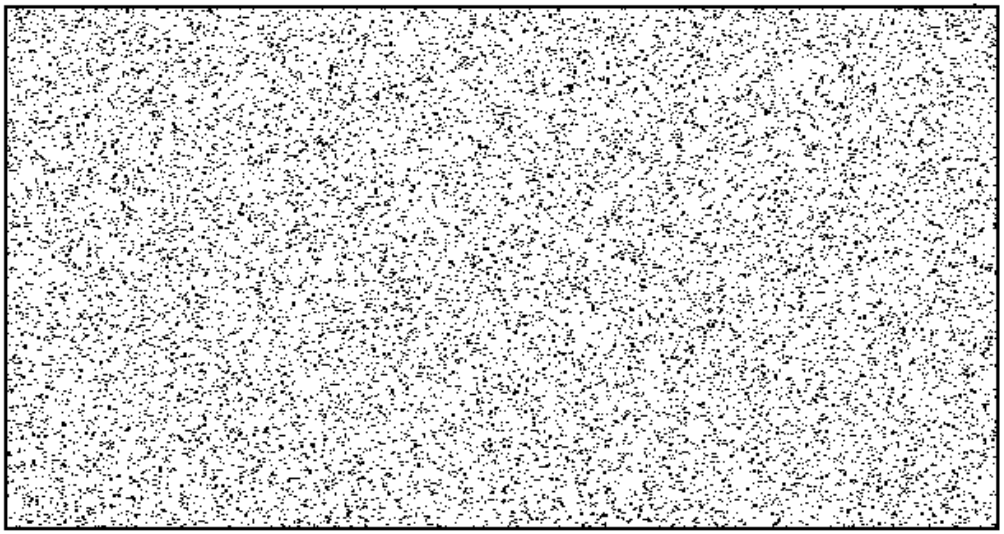


From Blockcodes to LDPC-Codes

Extensions to LDPC-Codes:

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

$\mathbf{H} =$

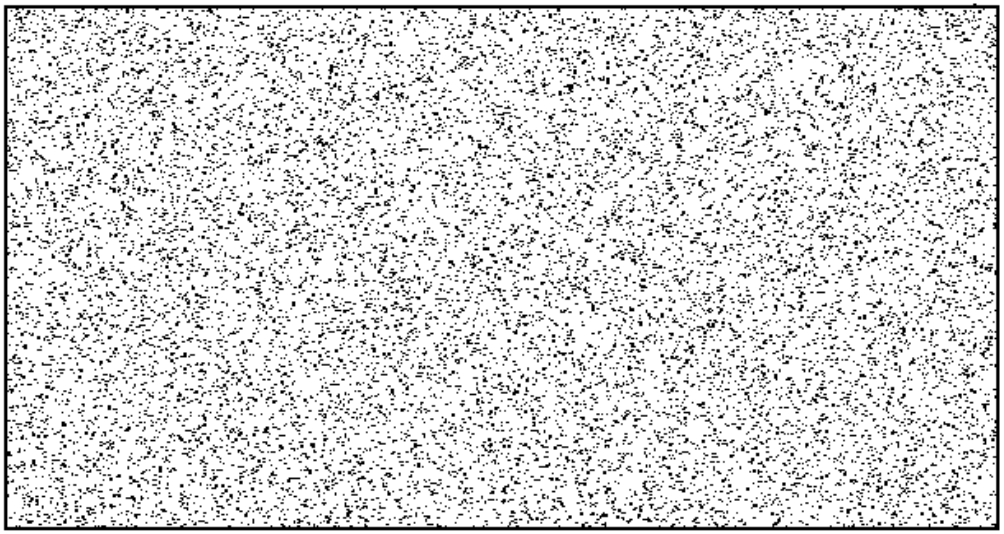


From Blockcodes to LDPC-Codes

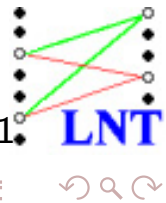
Extensions to LDPC-Codes:

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

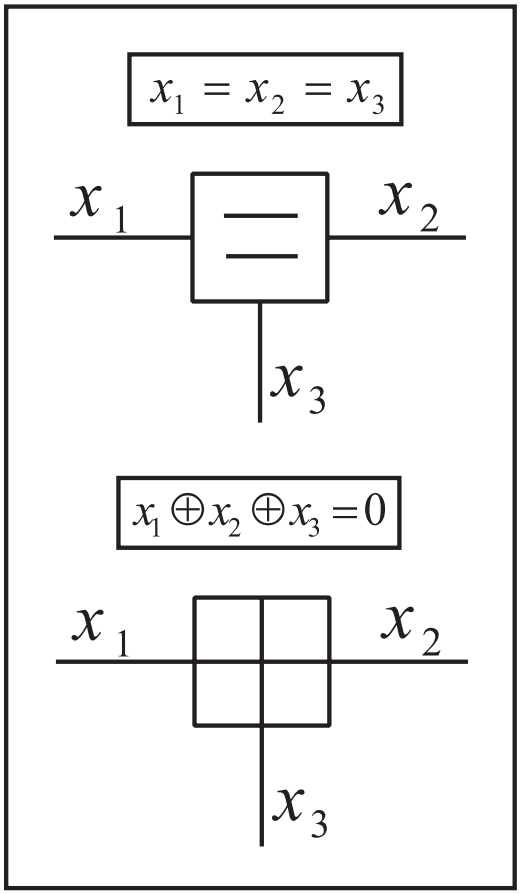
$\mathbf{H}$ =



- High encoding complexity arises from the high density in $\boldsymbol{H}_2^{-1}$



Graphical representation I



$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

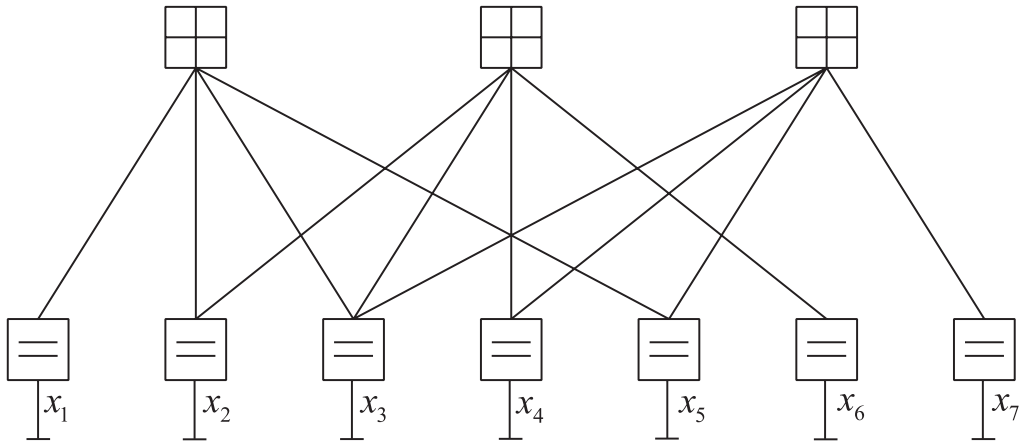


Figure: Graph of Hamming Code.

Graphical representation II

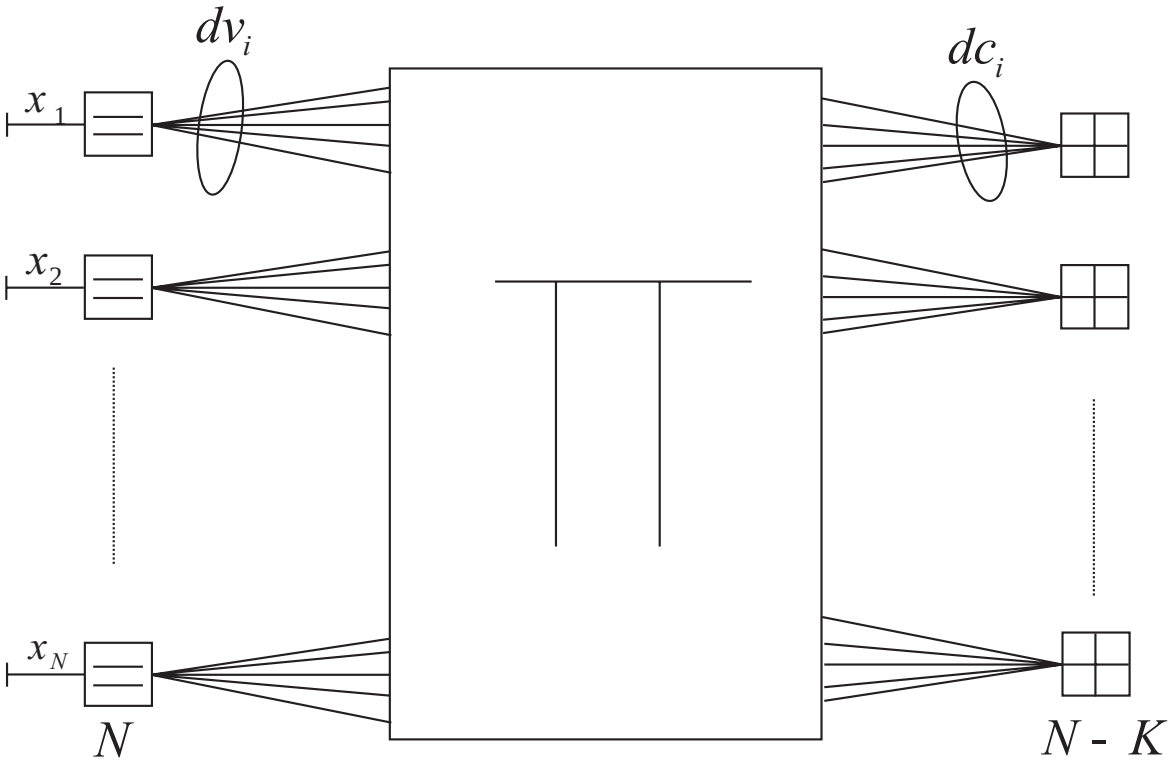


Figure: Graph of LDCP Codes.

Notation

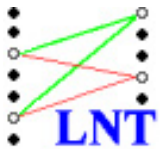
- dv_i degree of ith variable node
- Π edge interleaver
- dc_i degree of ith check node
- regular code:

$$dv_i = c_1 \quad \forall i$$

$$dc_i = c_2 \quad \forall i$$

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



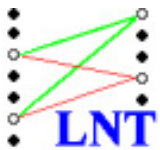
RA Code is a special LDPC Code

Practical interest due to low encoding complexity:

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

RA Code special structure:

$$\boldsymbol{H_2} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \Rightarrow \boldsymbol{H_2^{-1}} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$



RA-Encoding

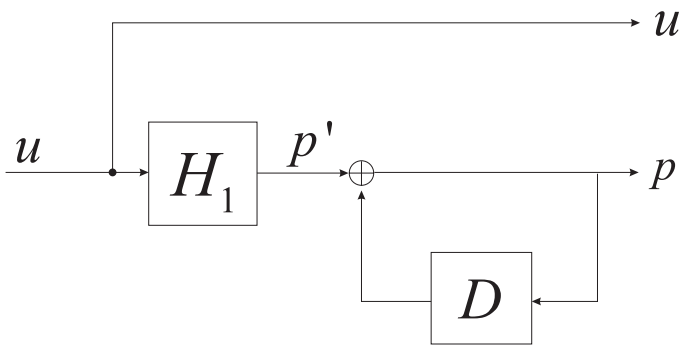
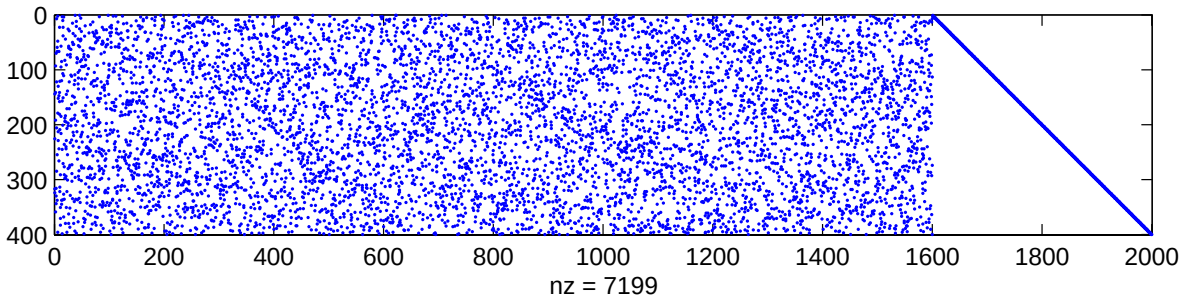


Figure: Block diagram of the simplified encoding.

Example Code: Intel proposal for IEEE802.16 rate 4/5 $N = 2000$



Graphical representation of RA Codes

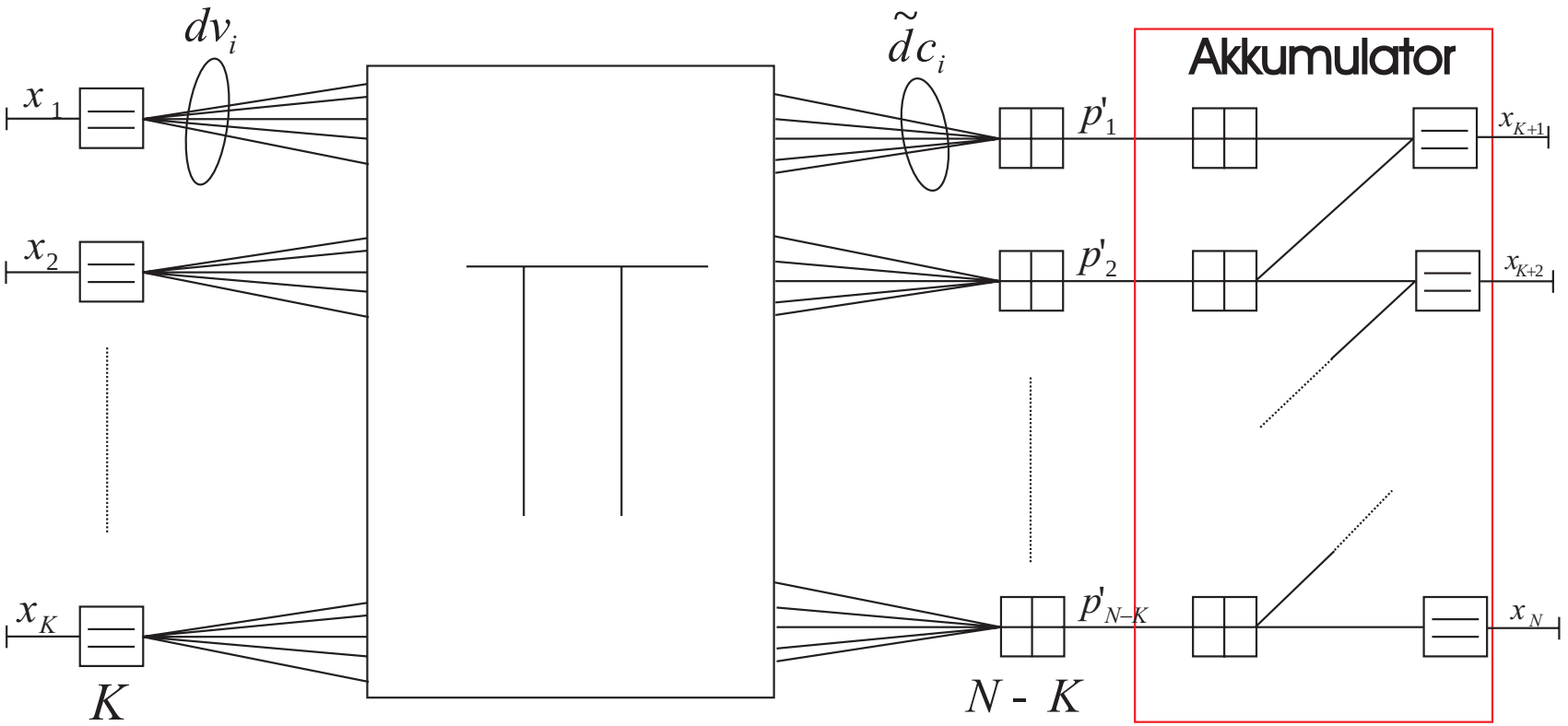
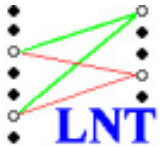
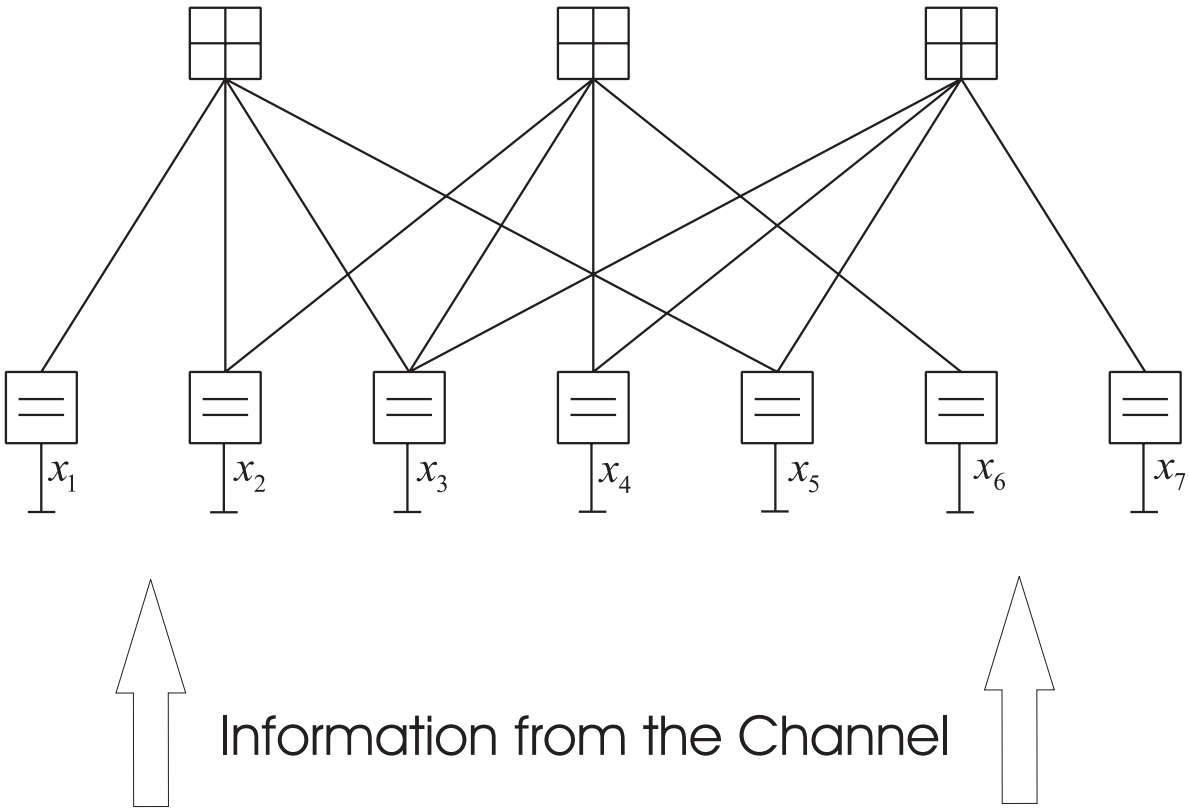


Figure: Graph of RA Codes.

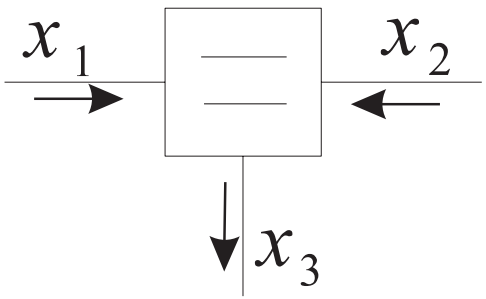
Outline

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



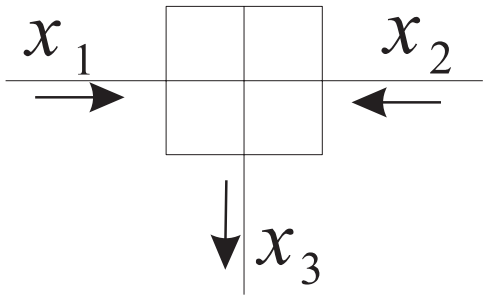

x_1
x_2
x_3
x_4
x_5
x_6
x_7
Information from the Channel

SPA for Block Codes



$$\Pr(x_3=0|y_3)=\Pr(x_1=0|y_1)\Pr(x_2=0|y_2)$$

$$\boxed{L_{x_3}=L_{x_2}+L_{x_3}}$$

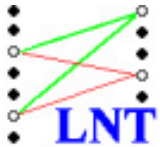


$$\Pr(x_3=0|y_3)=\Pr(x_1=0|y_1)\Pr(x_2=0|y_2)+\Pr(x_1=1|y_1)\Pr(x_2=1|y_2)$$

$$\boxed{\begin{aligned}L_{x_3}&=L_{x_1}\boxplus L_{x_2}\\&=2\tanh^{-1}(\tanh(L_{x_1}/2)\tanh(L_{x_2}/2))\end{aligned}}$$

SPA for LDPC Codes

- The SPA delivers exact a posteriori probabilities if the graph is cycle free.



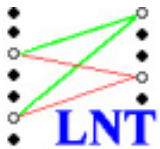
SPA for LDPC Codes

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

SPA for LDPC Codes

- The SPA delivers exact a posteriori probabilities if the graph is cycle free.
- LDPC graphs are not cycle free $\Rightarrow$ infinite message passing.
- Even in a fixpoint: no accurate probabilities
- We don't care. We just want the right codeword

$\Rightarrow$ "Apply the rules and hope for the best"

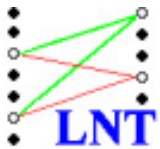


SPA for LDPC Codes

- The SPA delivers exact a posteriori probabilities if the graph is cycle free.
- LDPC graphs are not cycle free $\Rightarrow$ infinite message passing.
- Even in a fixpoint: no accurate probabilities
- We don't care. We just want the right codeword

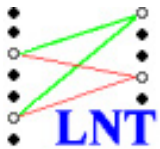
$\Rightarrow$ "Apply the rules and hope for the best"

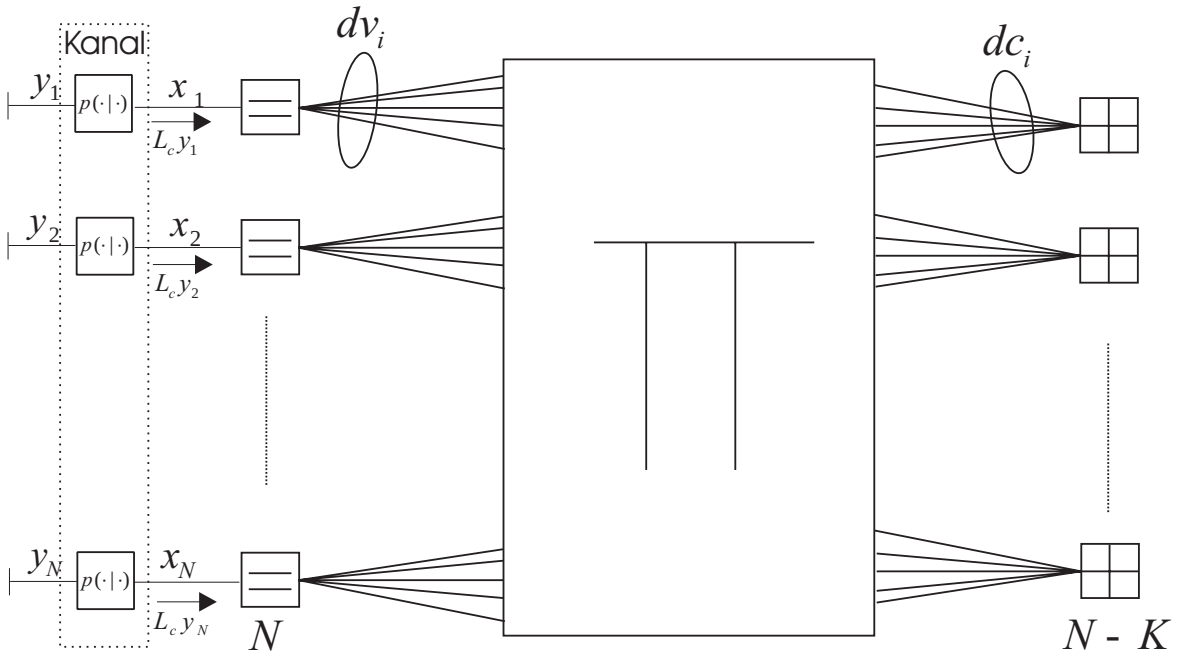
- In practice: It works!
  - timing schedule needed
  - restrictions to code design apply.



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

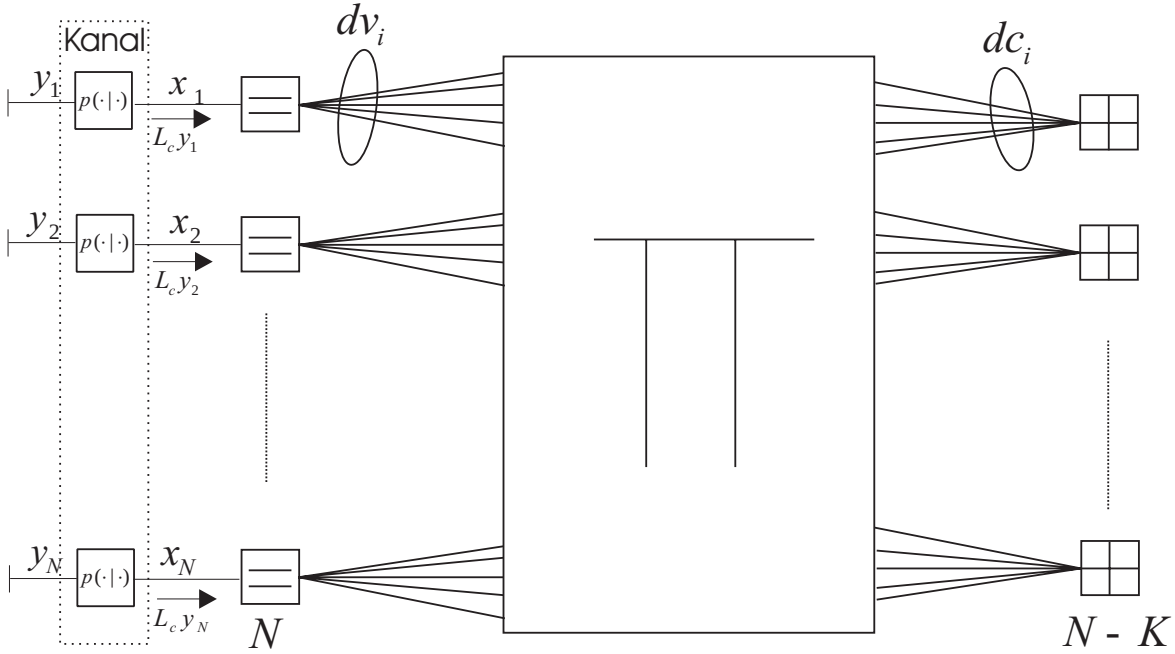




Update Rule Variable Nodes

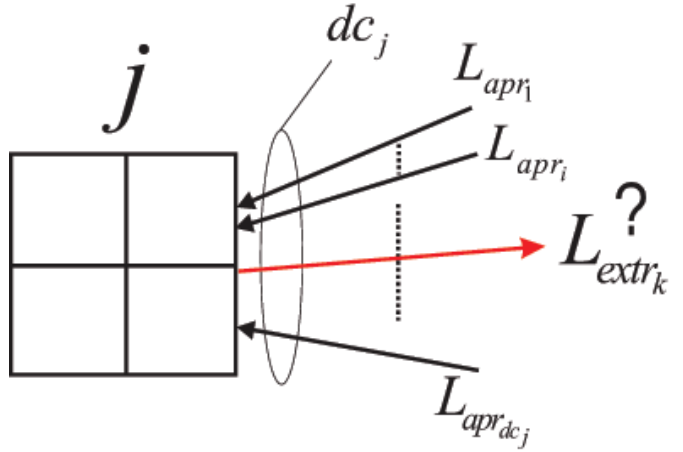
$$L_{intr_j} = L_c y_j + \sum_{i=1}^{dv_j} L_{apr_i} \quad \forall j = 1, ..N$$

$$L_{extr_i} = L_{intr_j} - L_{apr_i} \quad \forall i = 1, .., dv_i j = 1, ..N$$

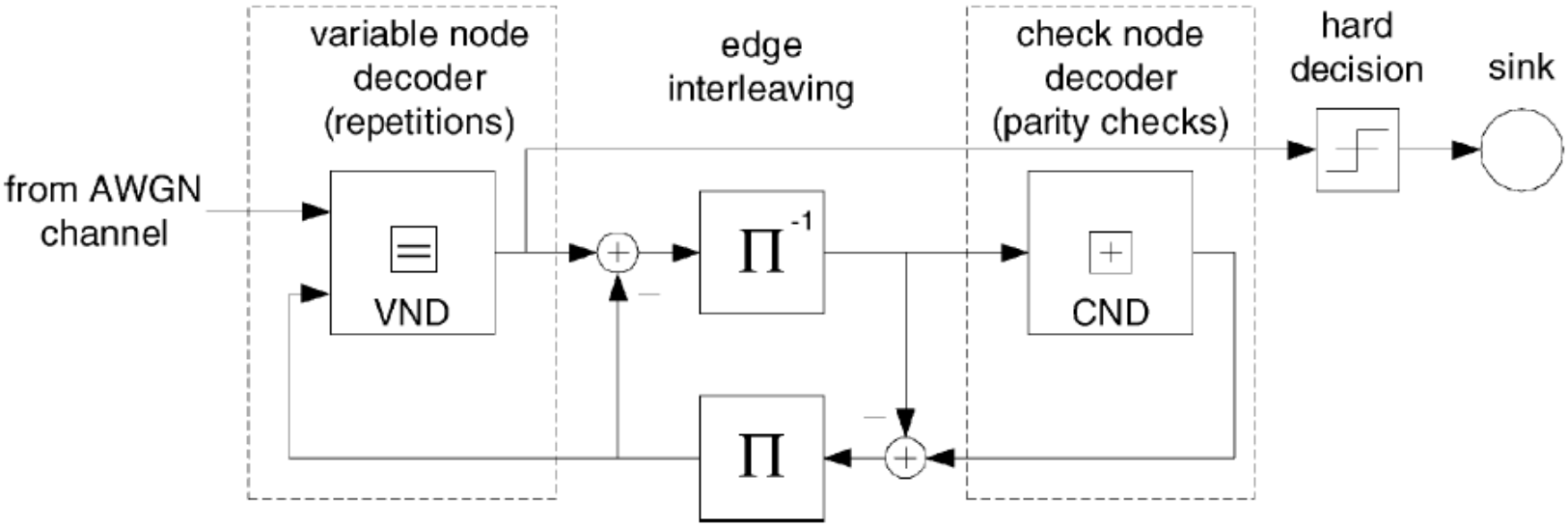


Update Rule Check Nodes

$$L_{extr_k} = \boxplus_{\substack{i=1 \\ i \neq k}}^{dc_j} L_{apr_i} \quad \forall k.$$

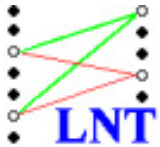


Turbo-Style view

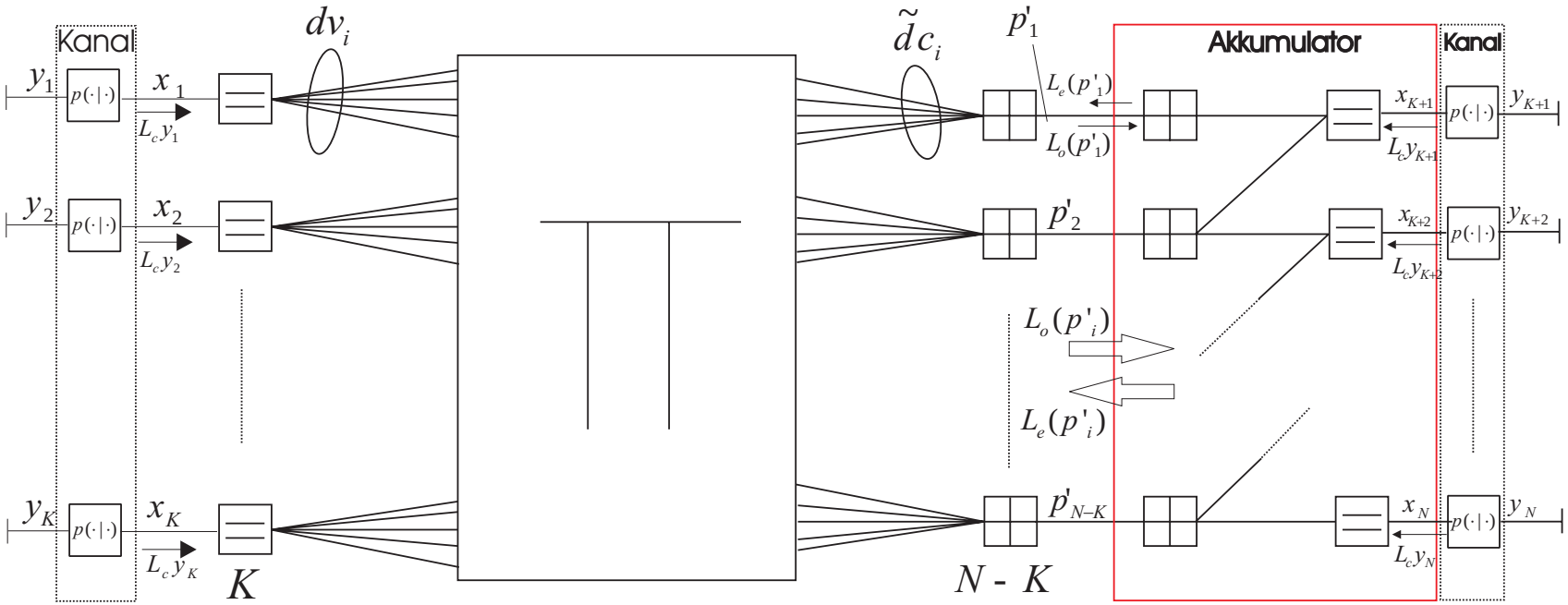


Outline

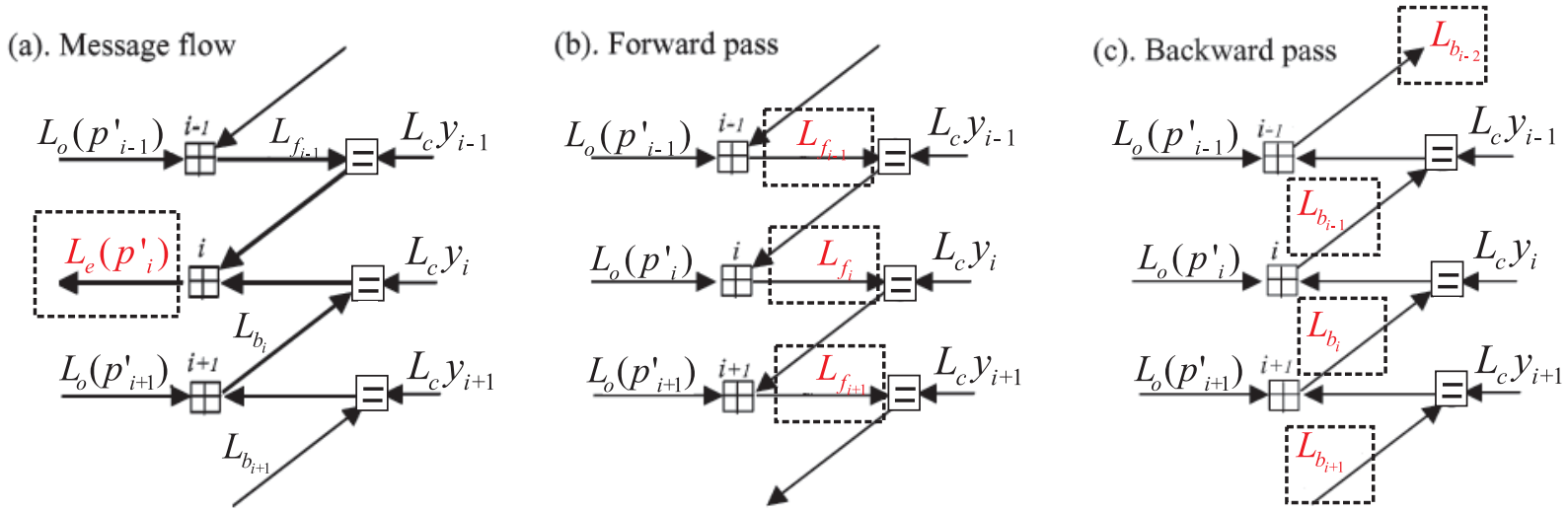
1. Introduction
2. Presentation of the Channel Codes
   - Low-Density Parity-Check Code
   - Repeat-Accumulate-Code
3. Decoding
   - Decoding using the Sum Product Algorithm
   - Decoding the LDPC-Code
   - Decoding the RA-Code
4. RA vs. LDPC
   - Performance
   - Complexity
5. Punctured RA-Codes
6.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



RA=LDPC+Accumulator



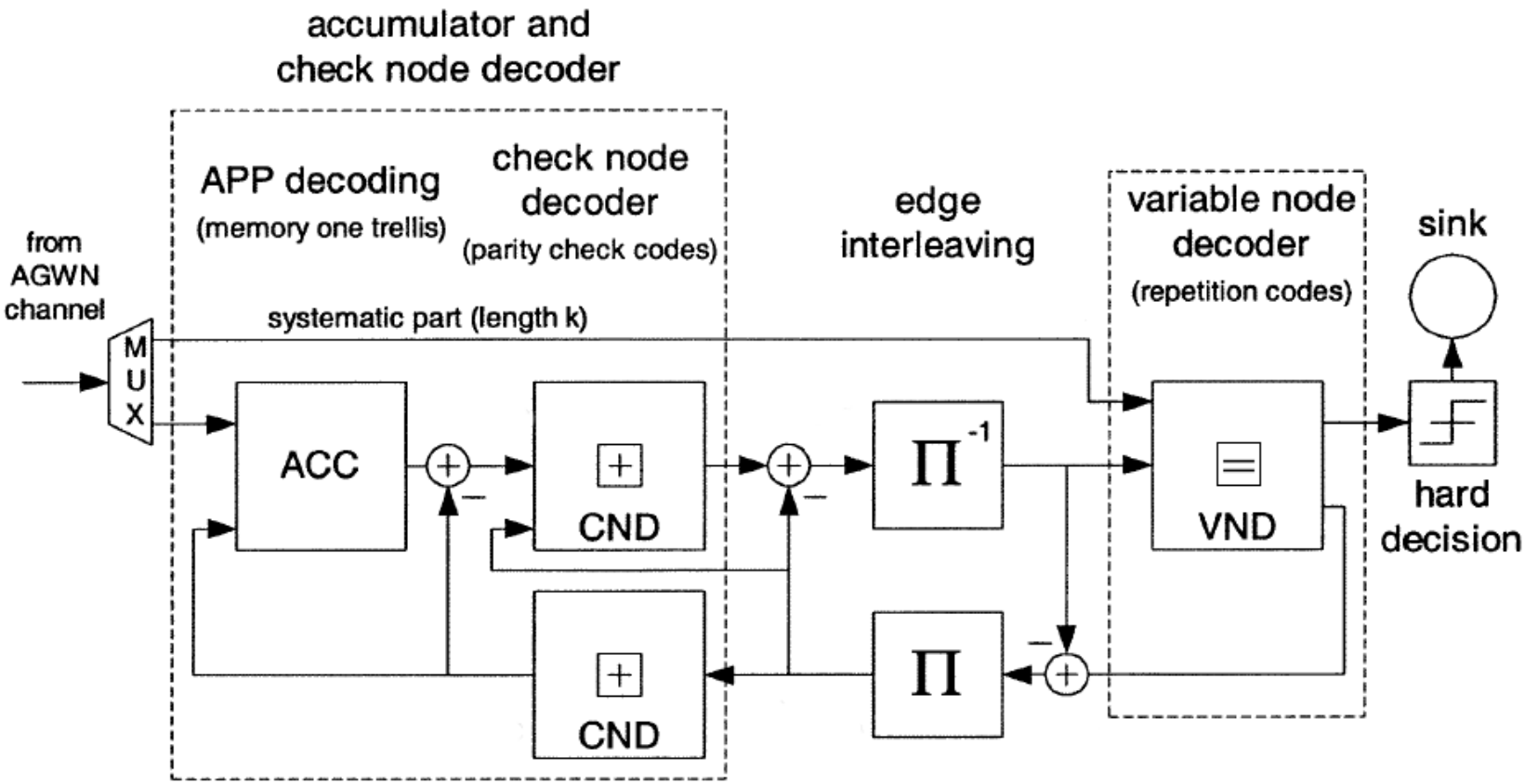
Efficient Accumulator decoding



Compute..

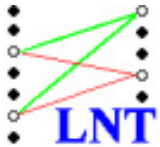
1. forward messages $L_{f_i} = (L_c y_{i-1} + L_{f_{i-1}}) \boxplus L_o(p'_i)$
2. backward messages $L_{b_i} = (L_c y_{i+1} + L_{b_{i+1}}) \boxplus L_o(p'_{i+1})$
3. outgoing messages $L_e(p'_i) = (L_{f_{i-1}} + L_c y_{i-1}) \boxplus (L_{b_i} + L_c y_i)$

Turbo view


accumulator and
check node decoder
APP decoding
(memory one trellis)
check node
decoder
(parity check codes)
edge
interleaving
variable node
decoder
(repetition codes)
sink
from
AGWN
channel
systematic part (length k)
M
U
X
ACC
CND
CND
VND
hard
decision

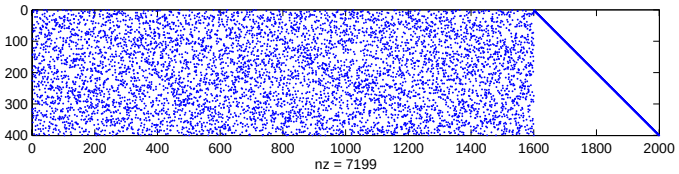
Outline

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

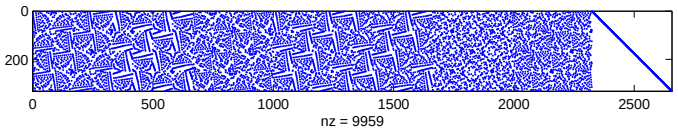


IEEE802.16 proposals

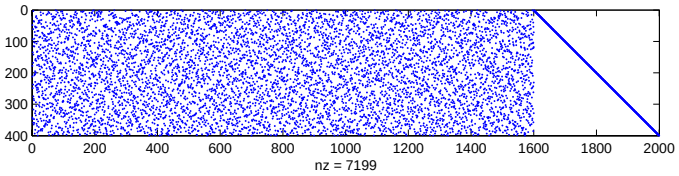
Intel R4/5, N=2000



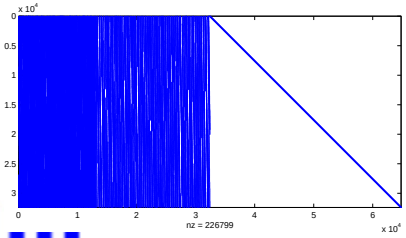
Nortel R8/9, N=2634



LG R1/2, N=2000



DVB-S2 Standards

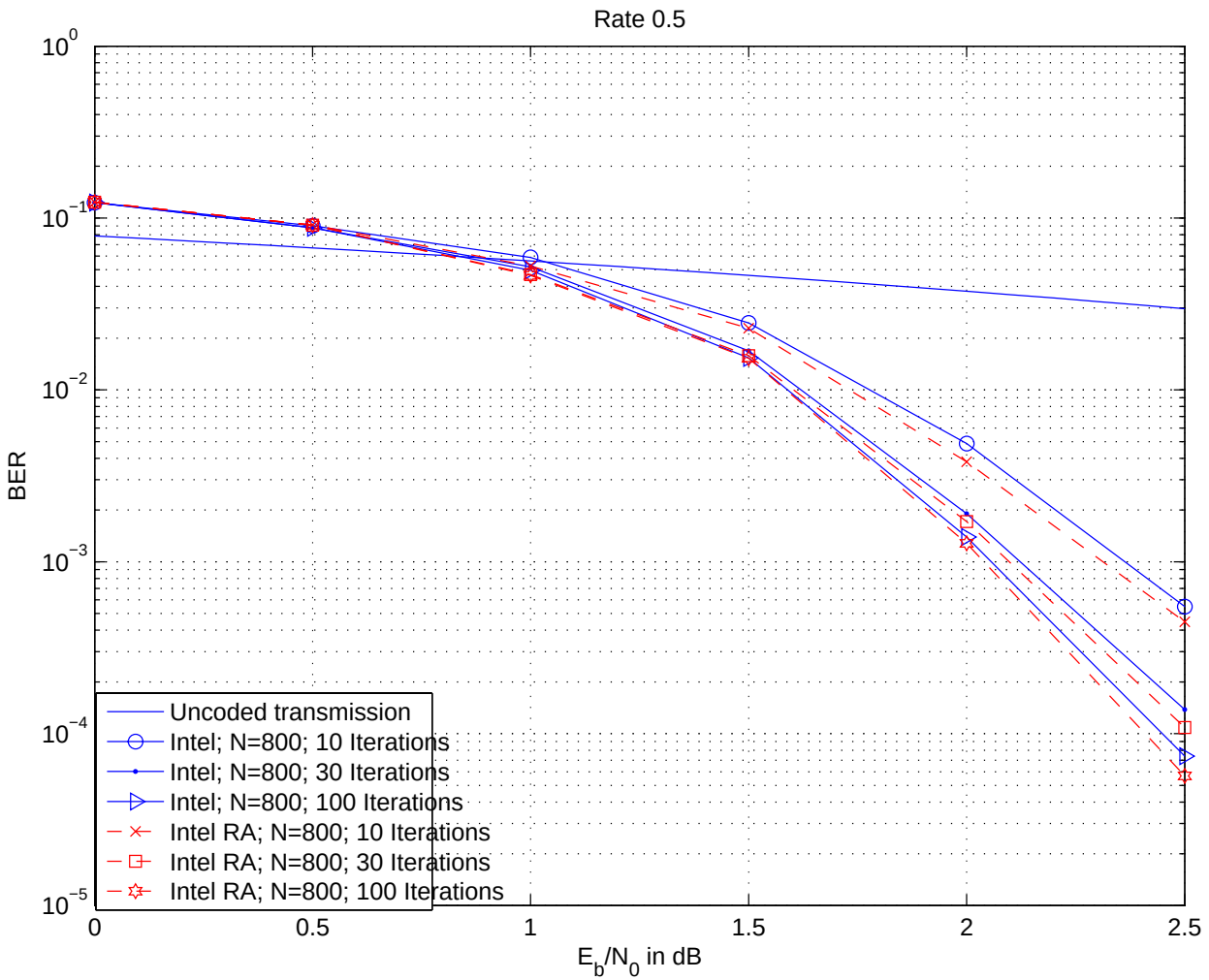


Motivation

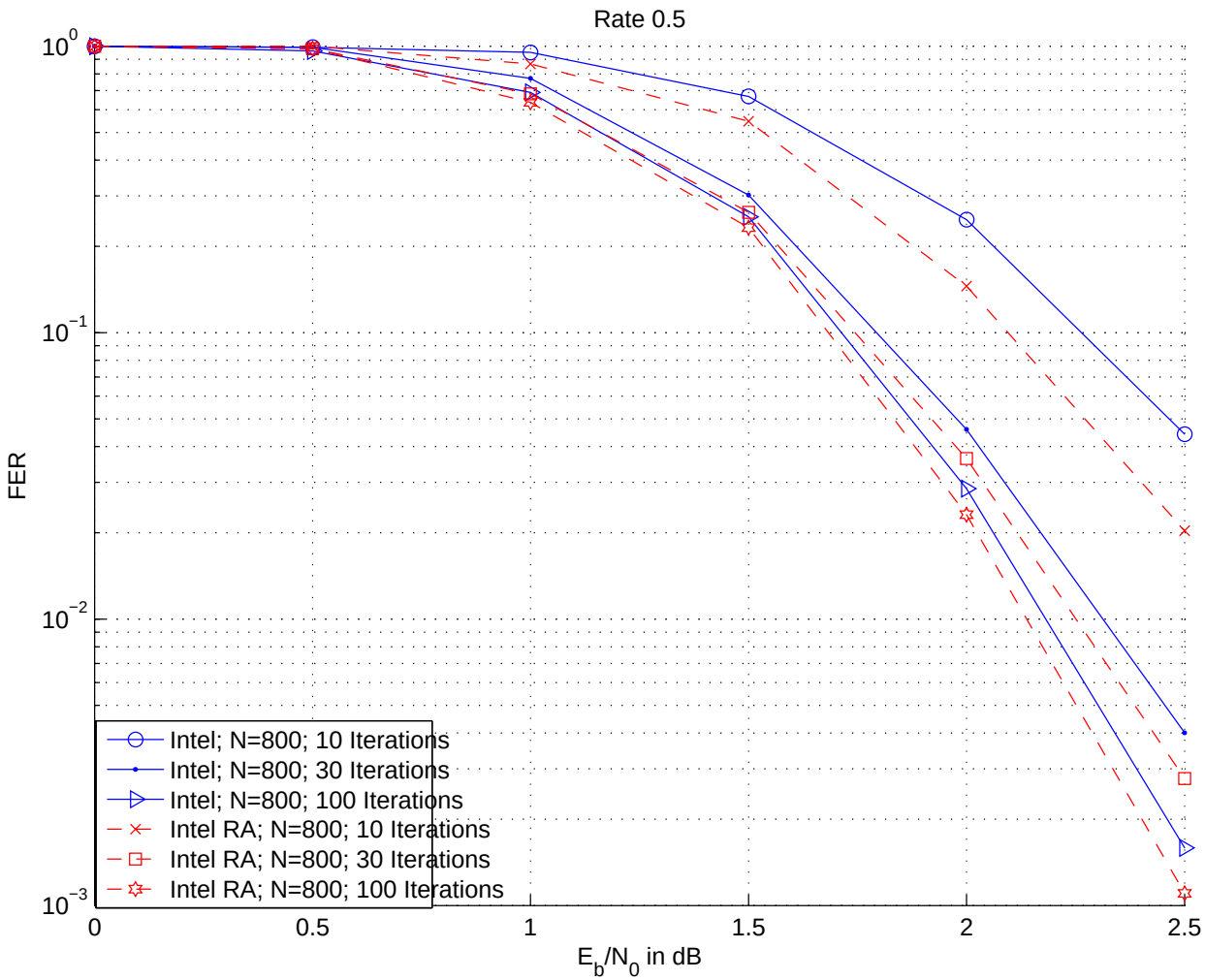
Nowadays applications demand low encoding complexity.

- ⇒ 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?

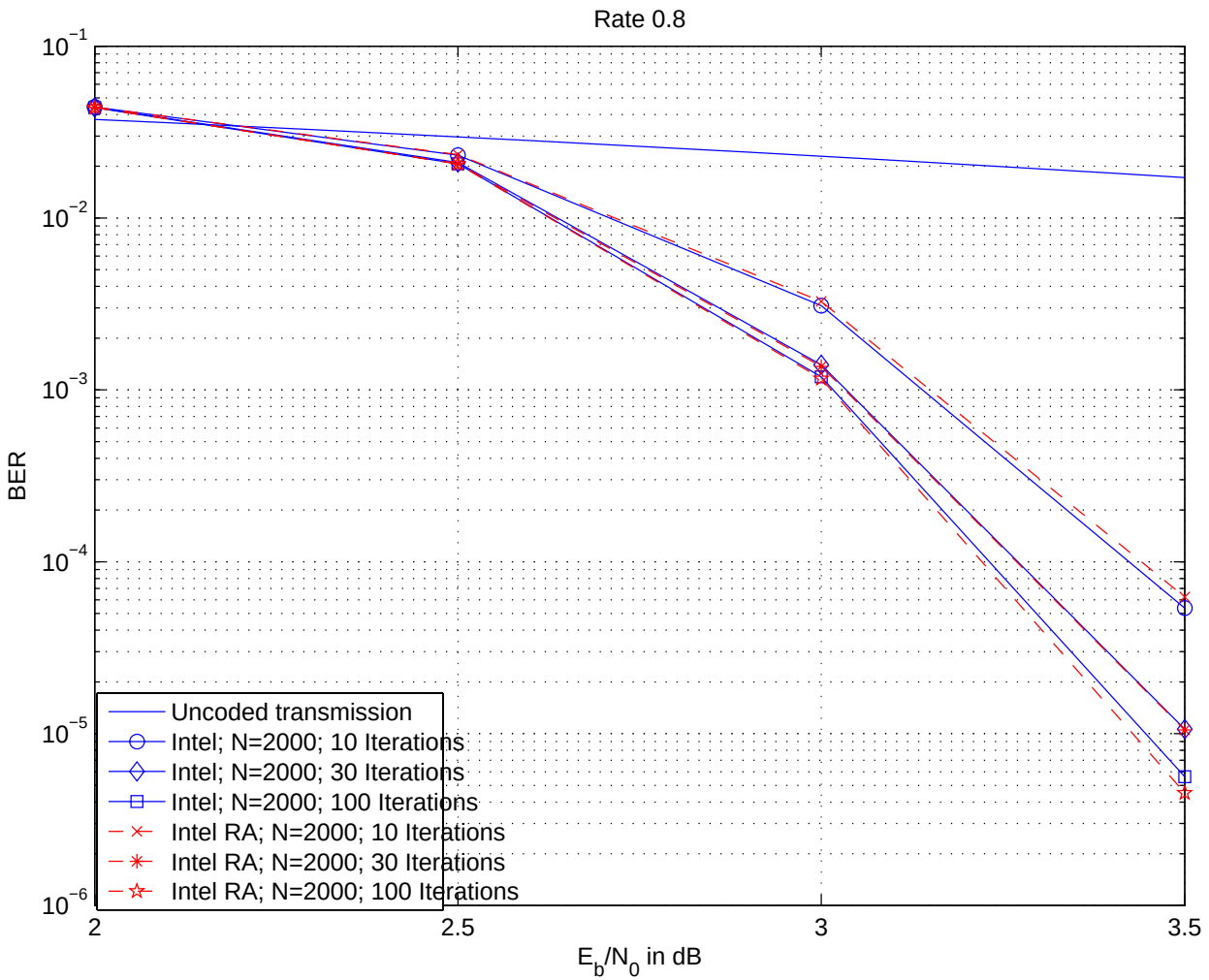
Intel proposal, Rate 1/2, N=800, BER



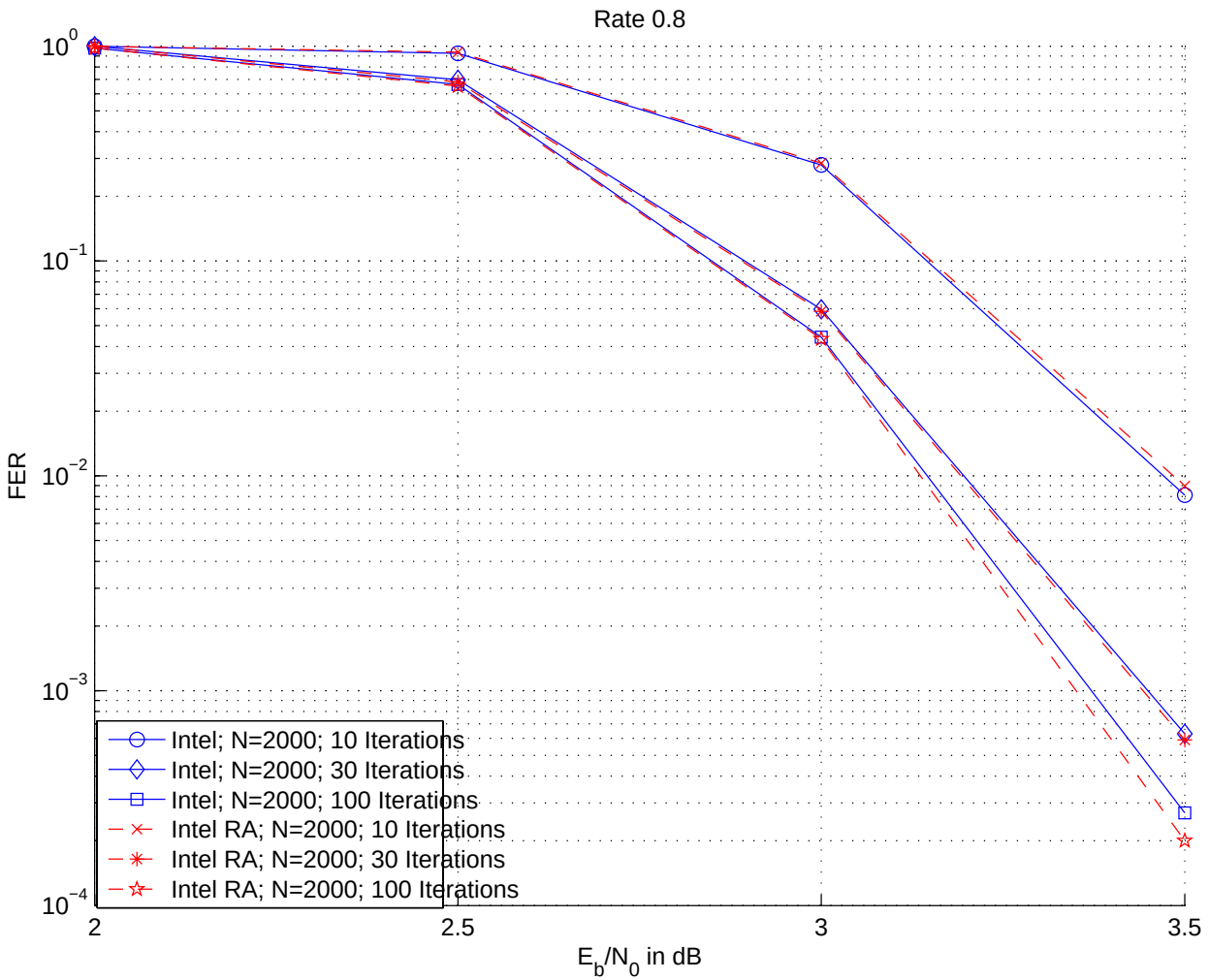
Intel proposal, Rate 1/2, N=800, FER



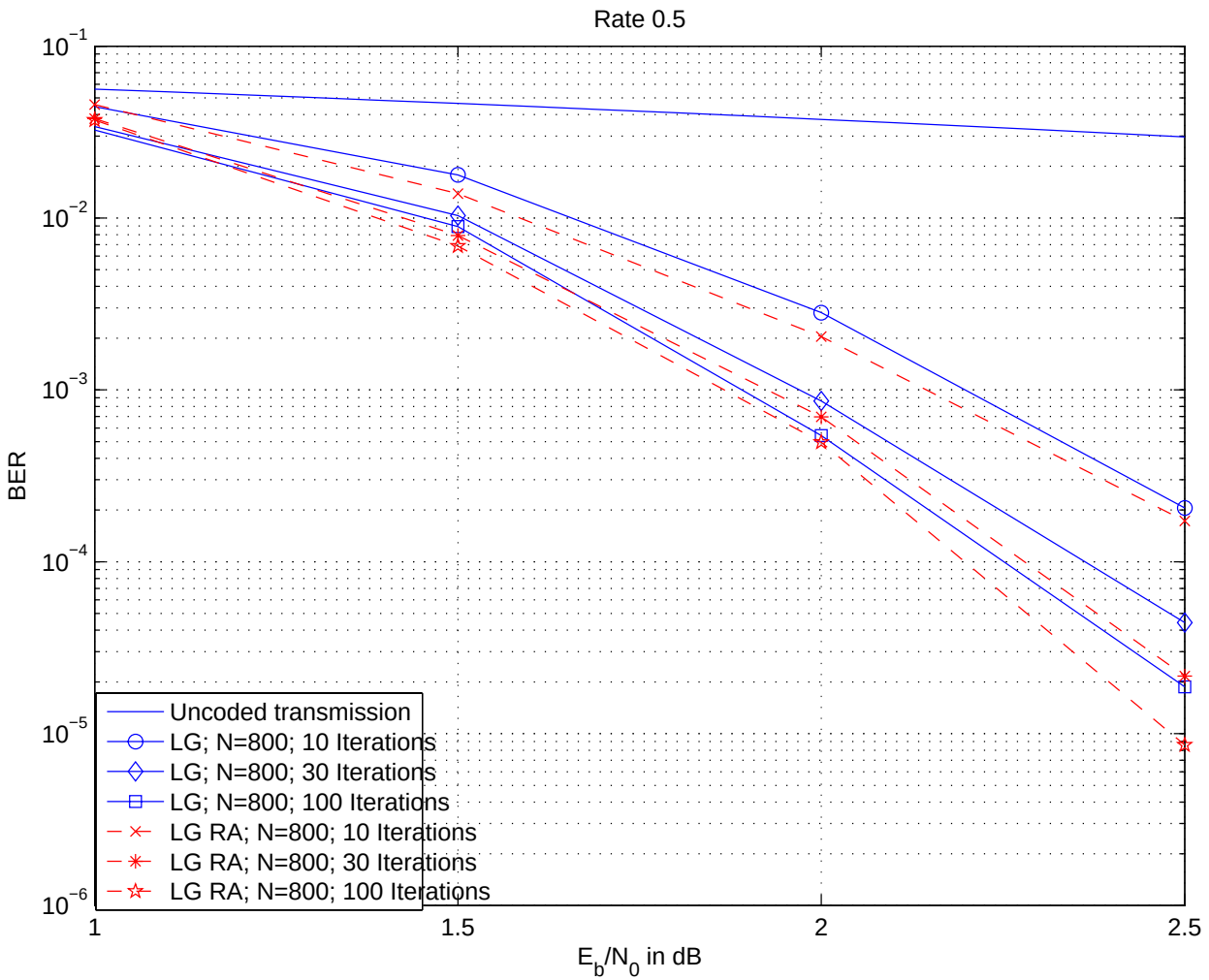
Intel proposal, Rate 4/5, N=2000, BER



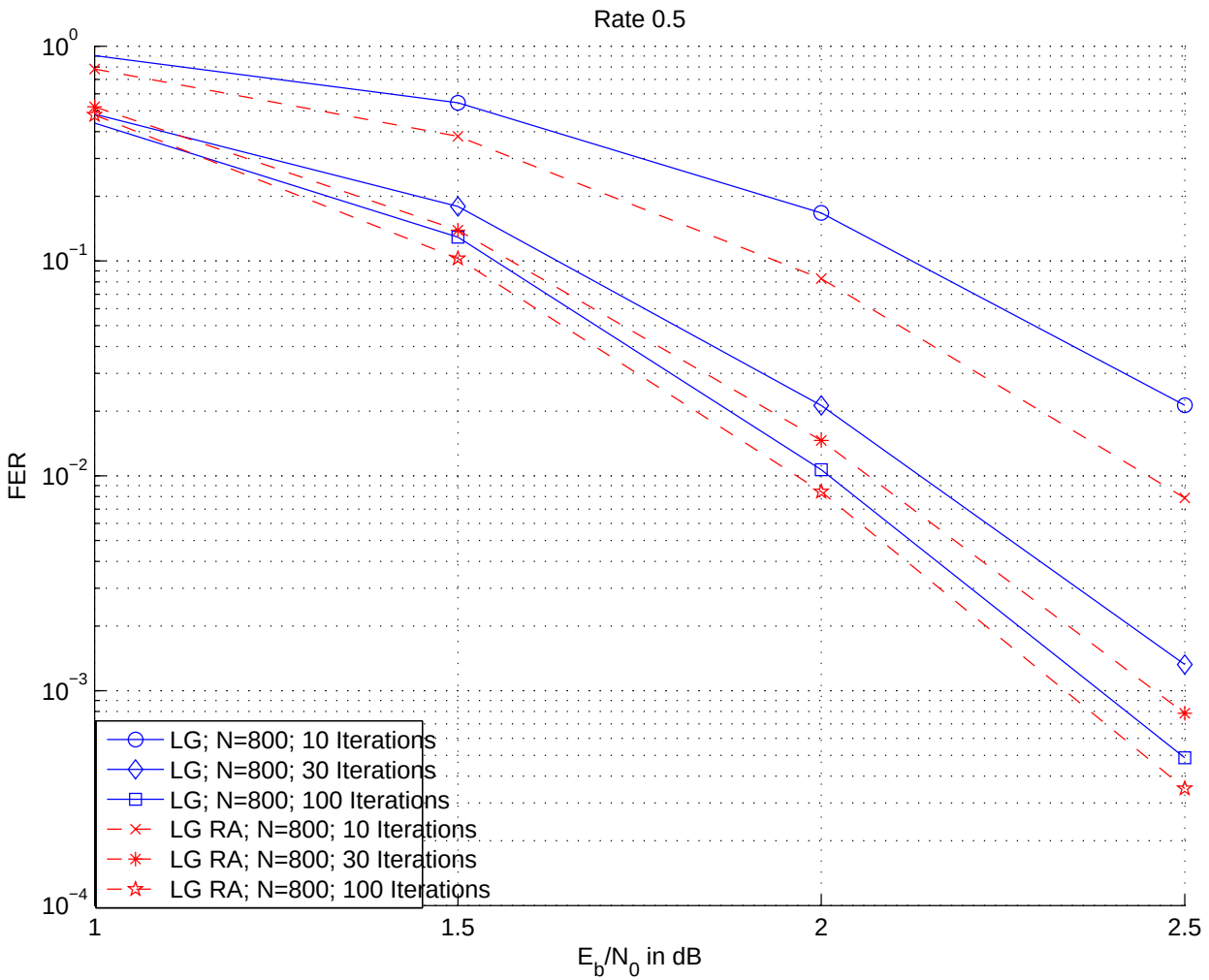
Intel proposal, Rate 4/5, N=2000, FER



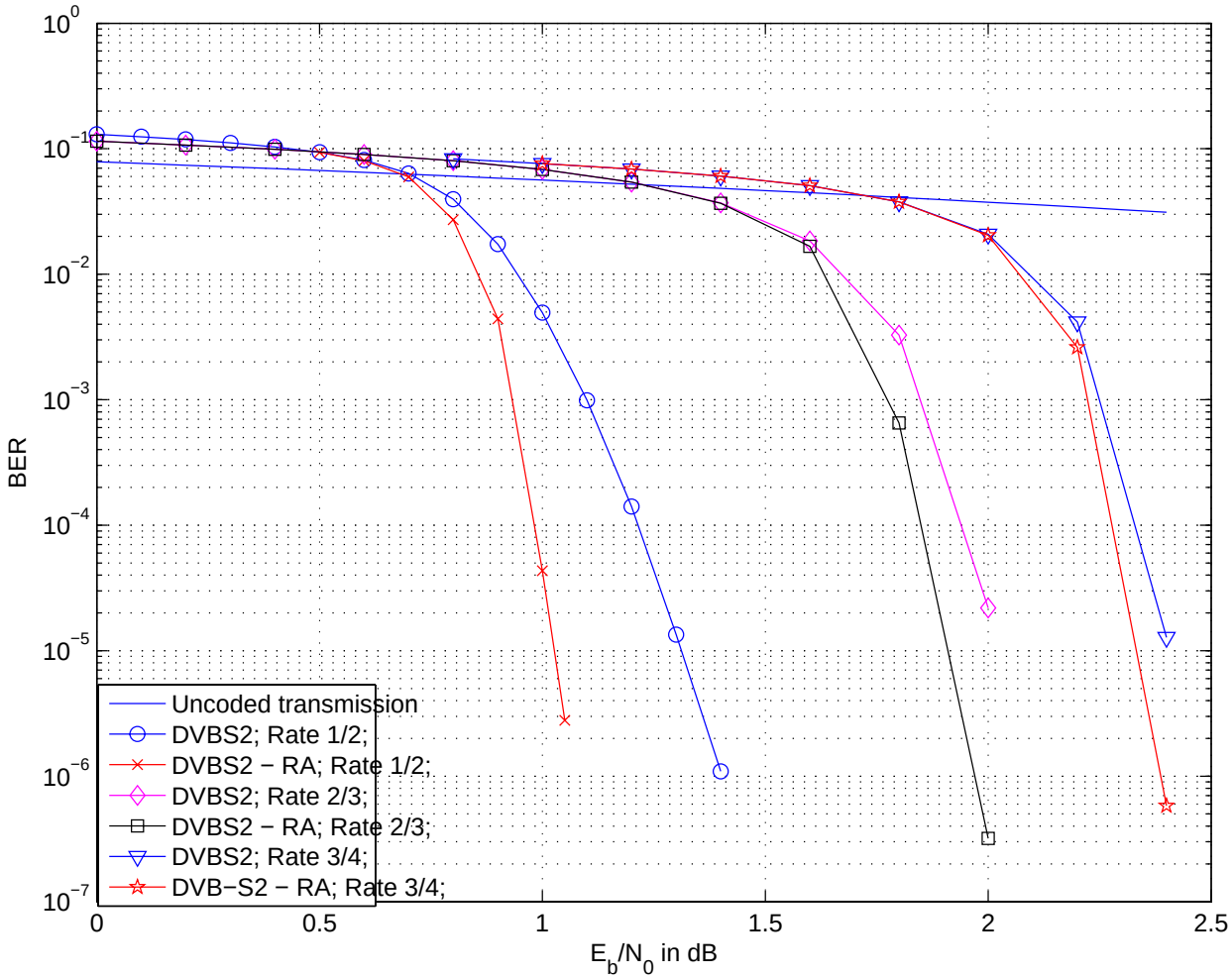
LG proposal, Rate 1/2, N=800, BER



LG proposal, Rate 1/2, N=800, FER

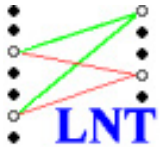


DVB-S2, N=64800, 20 Iterations



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?

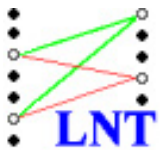
Let's count the numbers of operations for one iteration.

of Additions for LDPC decoding:

$$\text{For every variable node: } L_{intr_j} = L_c y_j + \sum_{i=1}^{dv_j} L_{apr_i}$$

$$\text{Extrinsic Information: } L_{extr_j} = Lch_i + \sum_{\substack{k=1 \\ k \neq j}}^{dv_i} L_{apriori_k}$$

$$\boxed{\#ADD_{LDPC} = 2\sum_{i=1}^{N} dv_i = 2NNZ(\boldsymbol{H}) := 2c}$$



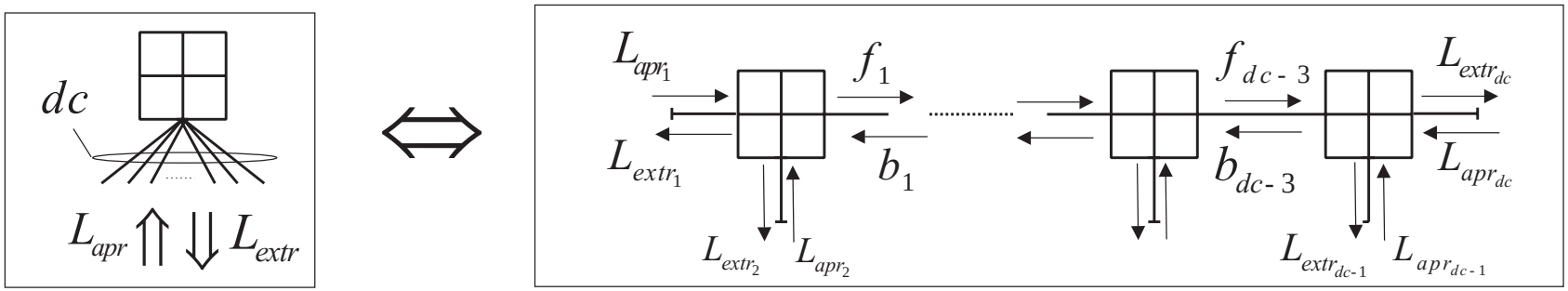


Figure: Efficient computation of checknode messages.

$\Rightarrow 3(dc_i - 2)$Boxplus operations per checknode

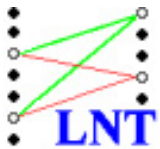
# of Boxplus operations for LDPC decoding

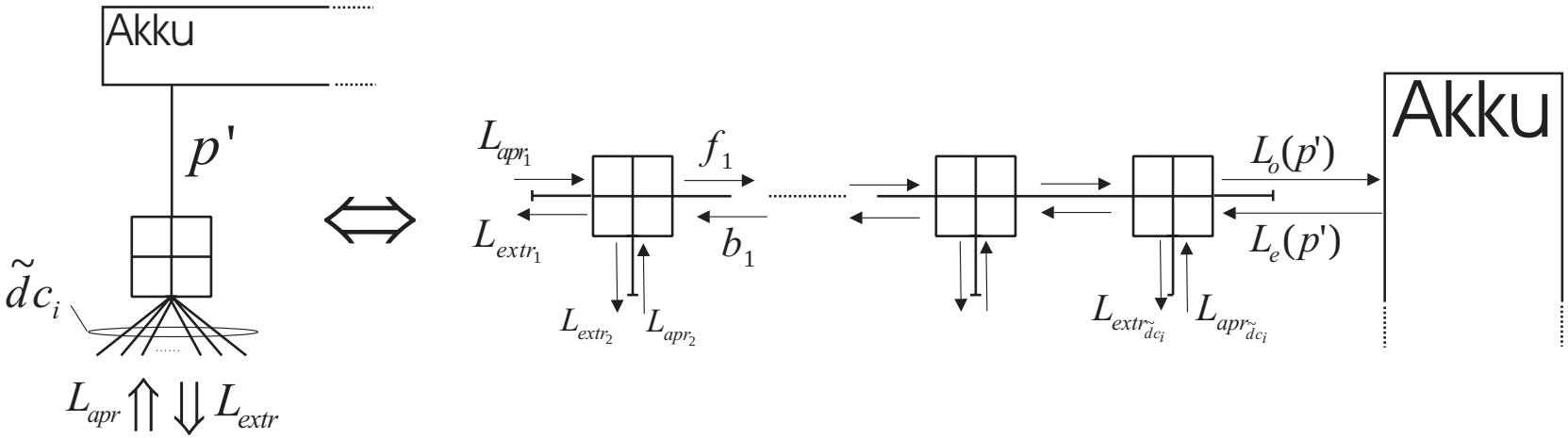
$$\#BXP_{LDPC} = 3 \sum_{i=1}^{N-K} (dc_i - 2) = 3c - 6(N - K)$$

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:

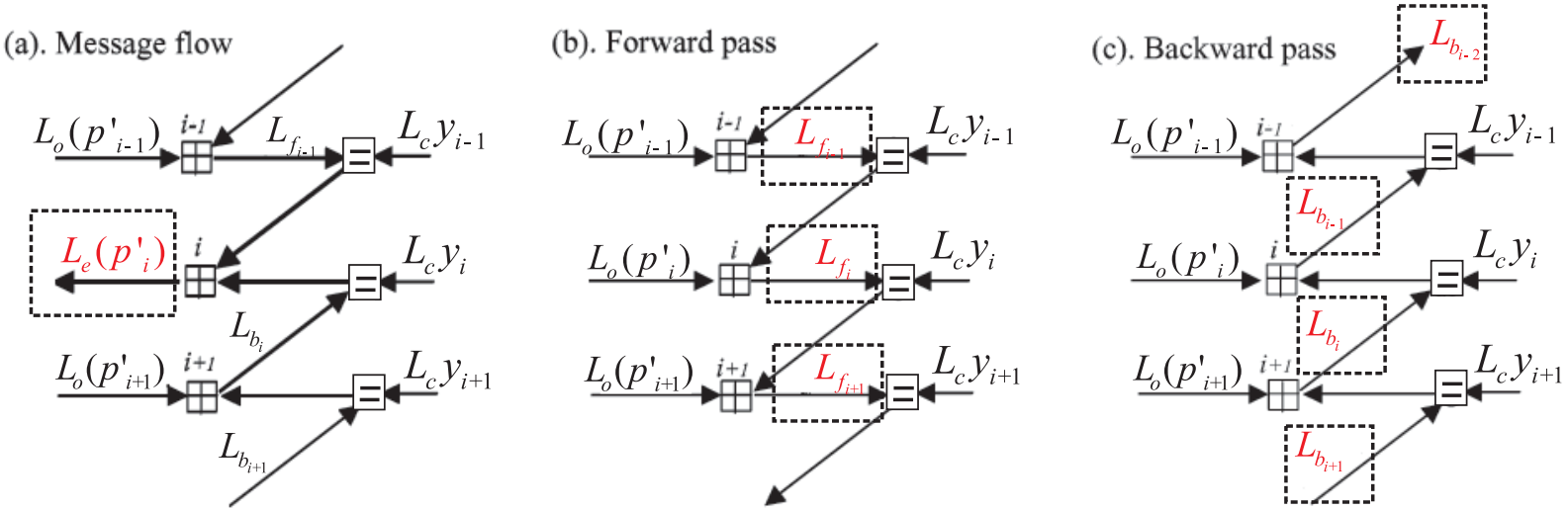
$$\#BXP_{RA1} = 3 \sum_{i=1}^{N-K} \left(\tilde{dc}_i - 1 \right)$$

Remember special structure of $\boldsymbol{H}$:

$$\tilde{dc}_1 = dc_1 - 1,$$
$$\tilde{dc}_i = dc_i - 2 \quad \forall i = 2, .., N-K$$

$$\Rightarrow \sum_{i=1}^{N-K} \left(3\tilde{dc}_i - 3\right) = 3 \sum_{i=1}^{N-K} \tilde{dc}_i - 3(N-K)$$
$$= 3dc_1 - 3 + 3 \sum_{i=2}^{N-K} dc_i - 6(N-K-1) - 3(N-K)$$
$$= 3 \sum_{i=1}^{N-K} dc_i - 9(N-K) + 3$$

$$\boxed{\#BXP_{RA1} = 3c - 9(N-K) + 3}$$



Computation	#Additions	#Bxpoperations	#total
Forward	N-K-2	N-K-2	2(N-K)-4
Backward	N-K-2	N-K-1	2(N-K)-3
Outgoing	2	(N-K-1)	N-K+1
total	2(N-K)-2	3(N-K)-4	5(N-K)-6

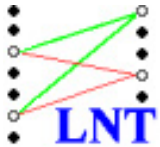
Table: Number of operations needed for accumulator

conclusion complexity analysis

Number of Operations of RA decoding

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

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



conclusion complexity analysis

Number of Operations of RA decoding

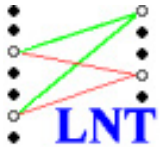
$$\#ADD_{RA} = 2\sum_{i=1}^{K} dv_i + 2(N-K) - 2$$

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

Number of Operations of LDPC decoding

$$\#ADD_{LDPC} = 2c$$

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



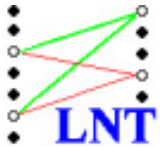
Number of RA BXP operations

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

Number of LDPC BXP operations

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

RA wins!

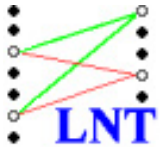


Number of RA Additions

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

Number of LDPC Additions

$$\#ADD_{LDPC} = 2c$$



Number of RA Additions

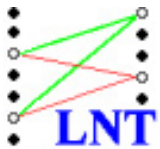
$$\#ADD_{RA} = 2\sum_{i=1}^{K} dv_i + 2(N-K) - 2$$

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

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

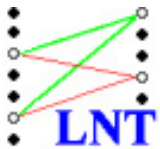
Number of LDPC Additions

$$\#ADD_{LDPC} = 2c$$

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

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

$$\#ADD_{LDPC} = 2c$$

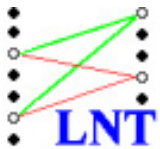
$$= 2\sum_{i=1}^{K} dv_i + 4(N-K) - 2$$

Remember special structure:

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

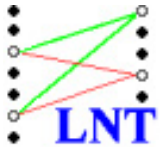
$$\Rightarrow \frac{\#ADD_{RA}}{\#ADD_{LDPC}} < 1 \quad \forall R < 1$$

RA wins!



Result of complexity analysis

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

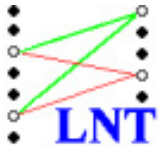


Result of complexity analysis

Independent of the Rate and independent of the entries in the Parity-Check Matrix $\boldsymbol{H}$: 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:

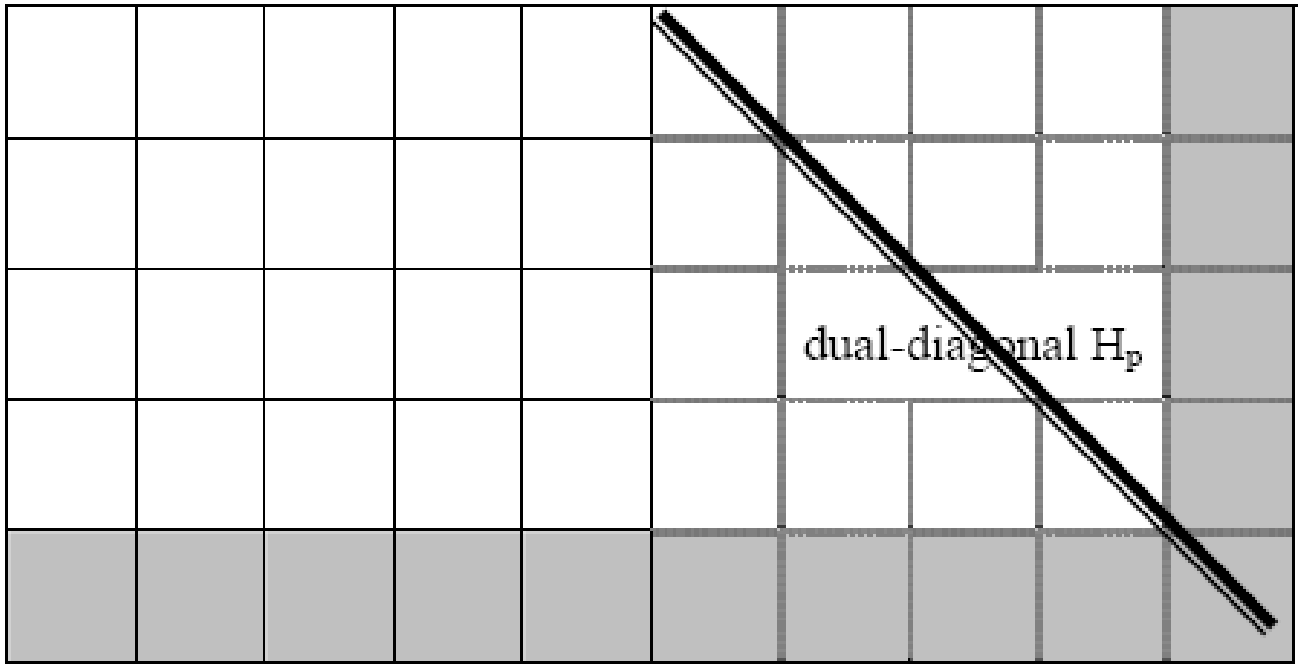
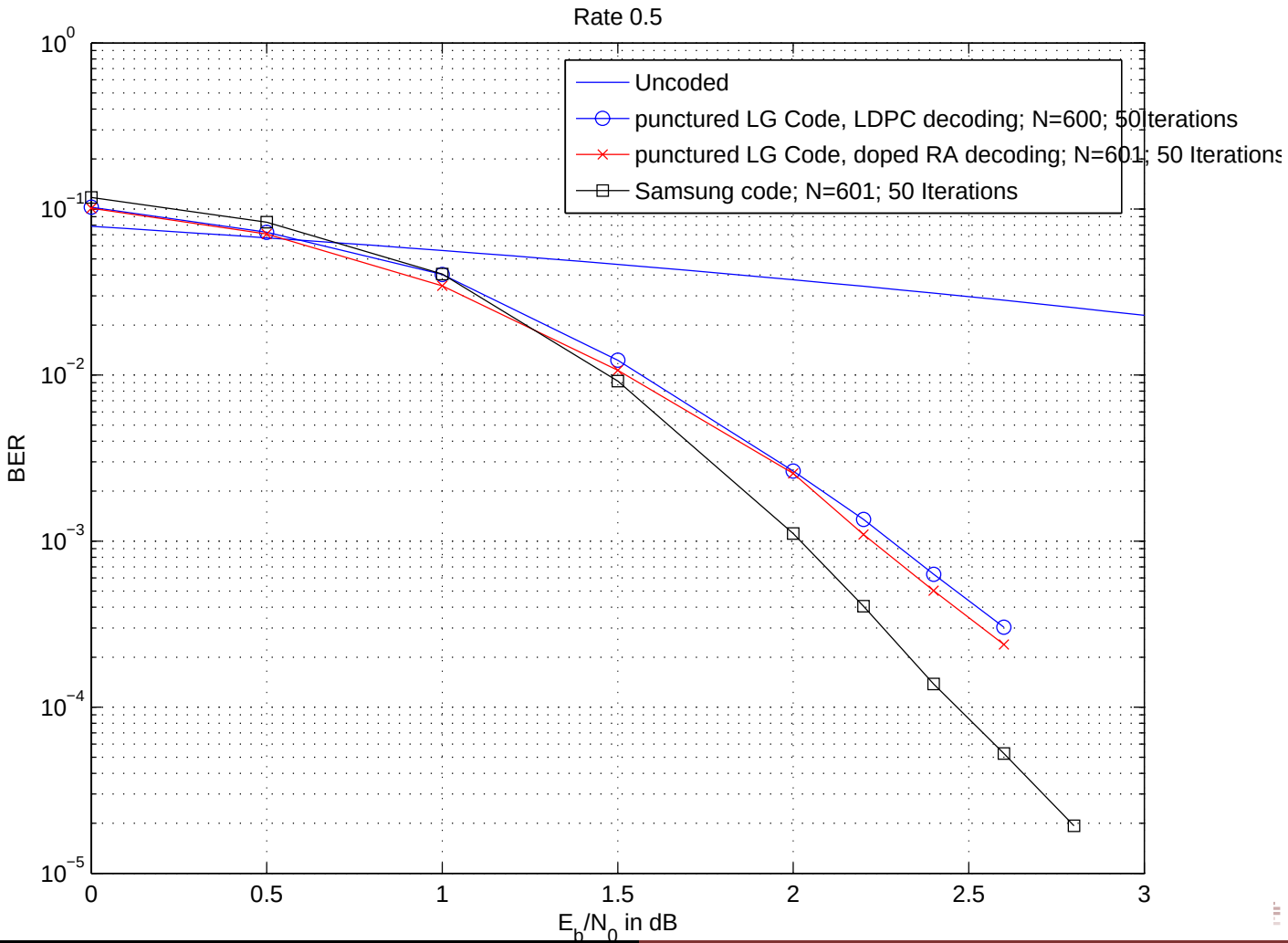


Figure 4. An example of reducing parity bits (shaded area is reduced)

leads to bad results



Maybe we can do better?

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

$$\boldsymbol{Hx} = [\boldsymbol{H_1}|\boldsymbol{H_2}]\boldsymbol{x} = \boldsymbol{H_1 u} + \boldsymbol{H_2 p} = \boldsymbol{I p'} + \boldsymbol{H_2 p} = \mathbf{0}.$$

$$\begin{bmatrix} p'_1 \\ p'_2 \\ p'_3 \\ p'_4 \\ \vdots \\ p'_{N-K} \end{bmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{bmatrix} x_{K+1} \\ x_{K+2} \\ x_{K+3} \\ x_{K+4} \\ \vdots \\ x_N \end{bmatrix} = \mathbf{0}.$$

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

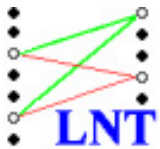
$$x_{N-99} = x_{N-100} \oplus p'_{N-K-99}$$
$$x_{N-98} = x_{N-99} \oplus p'_{N-K-98}$$
$$\dots$$

Decoding

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

$\Rightarrow$ First approach: preprocessing.

$$L(p_i) = \left(\sum_{j=1}^{K} \boxplus H_{i,j} Lcy_j \right) \boxplus L(p_{i-1})$$



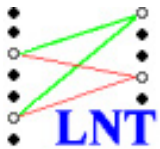
Decoding

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

$\Rightarrow$ First approach: preprocessing.

$$L(p_i) = \left(\sum_{j=1}^{K} \boxplus H_{i,j} L c y_j \right) \boxplus L(p_{i-1})$$

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.



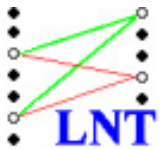
New puncturing scheme: puncture every second bit.

$\Rightarrow$ punctured Bit can be recovered with information from transmitted Bit.

$\Rightarrow$ Preprocessing + LDPC decoding:

$$L(p_{2i}) = \left(\sum_{j=1}^{K} \boxplus H_{i,j} L_c y_j \right) \boxplus L(p_{2i-1})$$

$$L_c y_{K+2i} = \left(\sum_{j=1}^{K} \boxplus H_{i,j} L_c y_j \right) \boxplus L_c y_{K+2i-1}$$



New puncturing scheme: puncture every second bit.

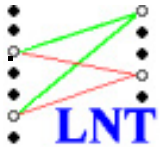
$\Rightarrow$ punctured Bit can be recovered with information from transmitted Bit.

$\Rightarrow$ Preprocessing + LDPC decoding:

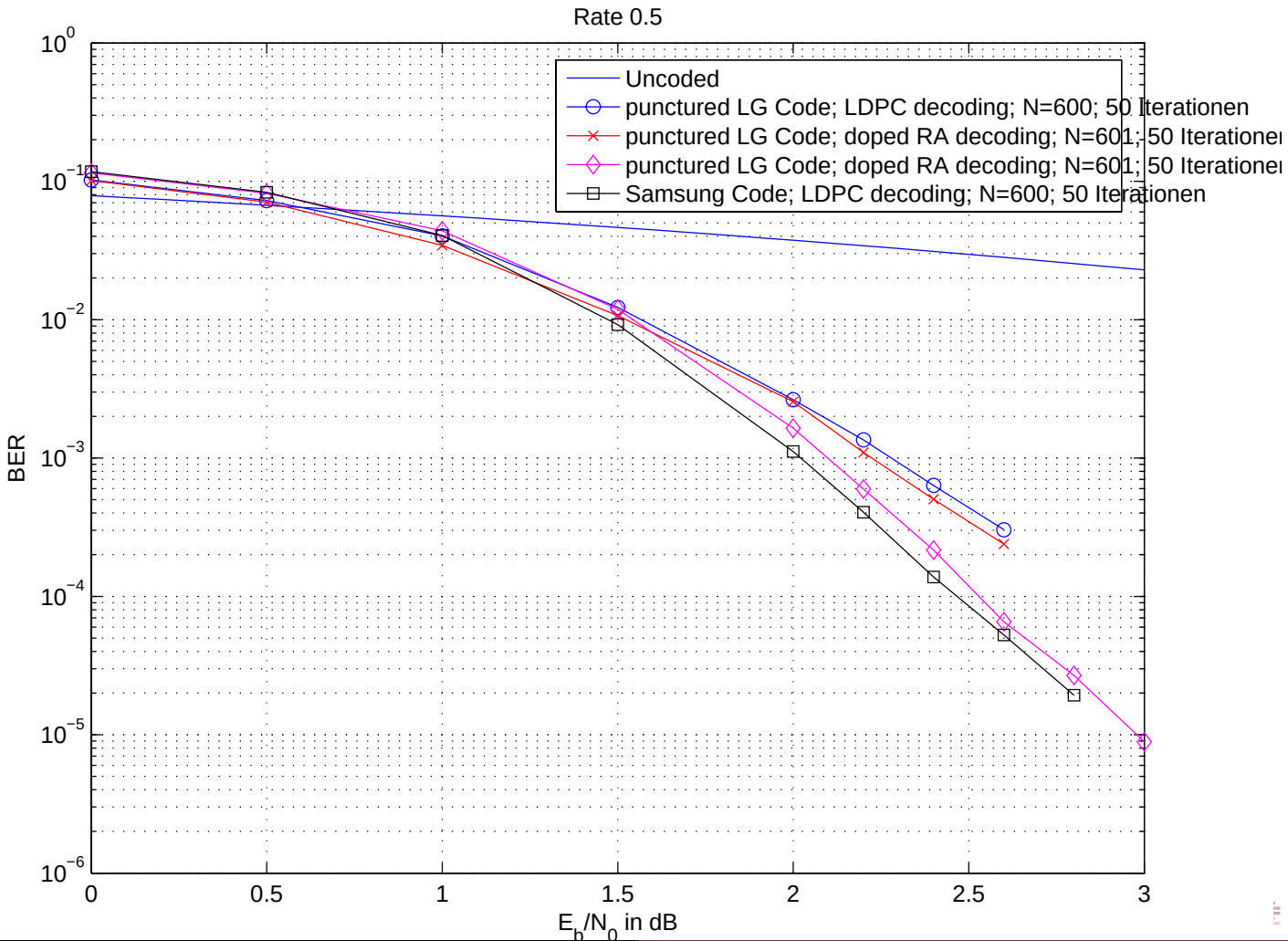
$$L(p_{2i}) = \left(\sum_{j=1}^{K} \boxplus H_{i,j} L_c y_j \right) \boxplus L(p_{2i-1})$$

$$L_c y_{K+2i} = \left(\sum_{j=1}^{K} \boxplus H_{i,j} L_c y_j \right) \boxplus L_c y_{K+2i-1}$$

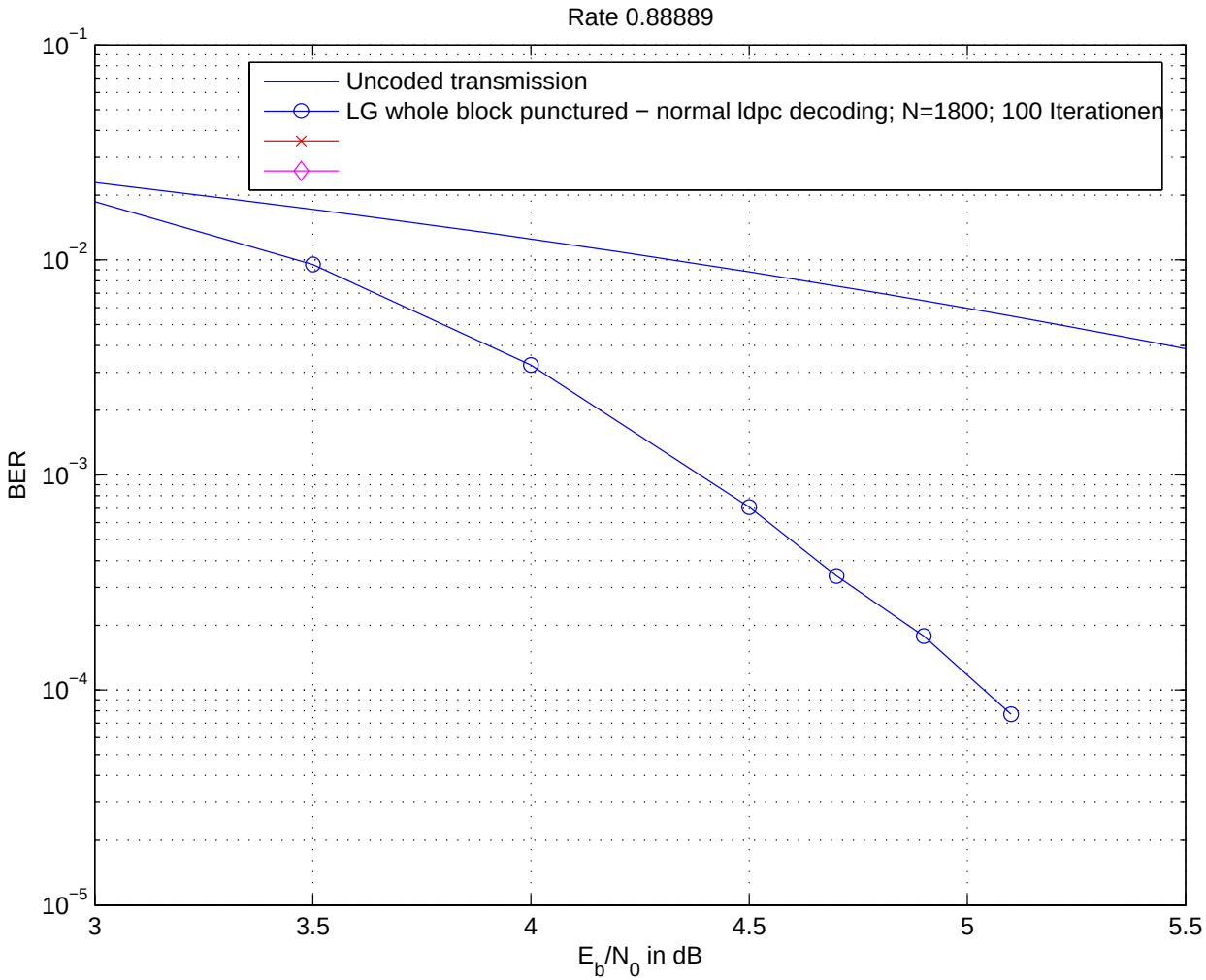
$\Rightarrow$ ..or "doped" RA decoding (preprocessing done automatically)



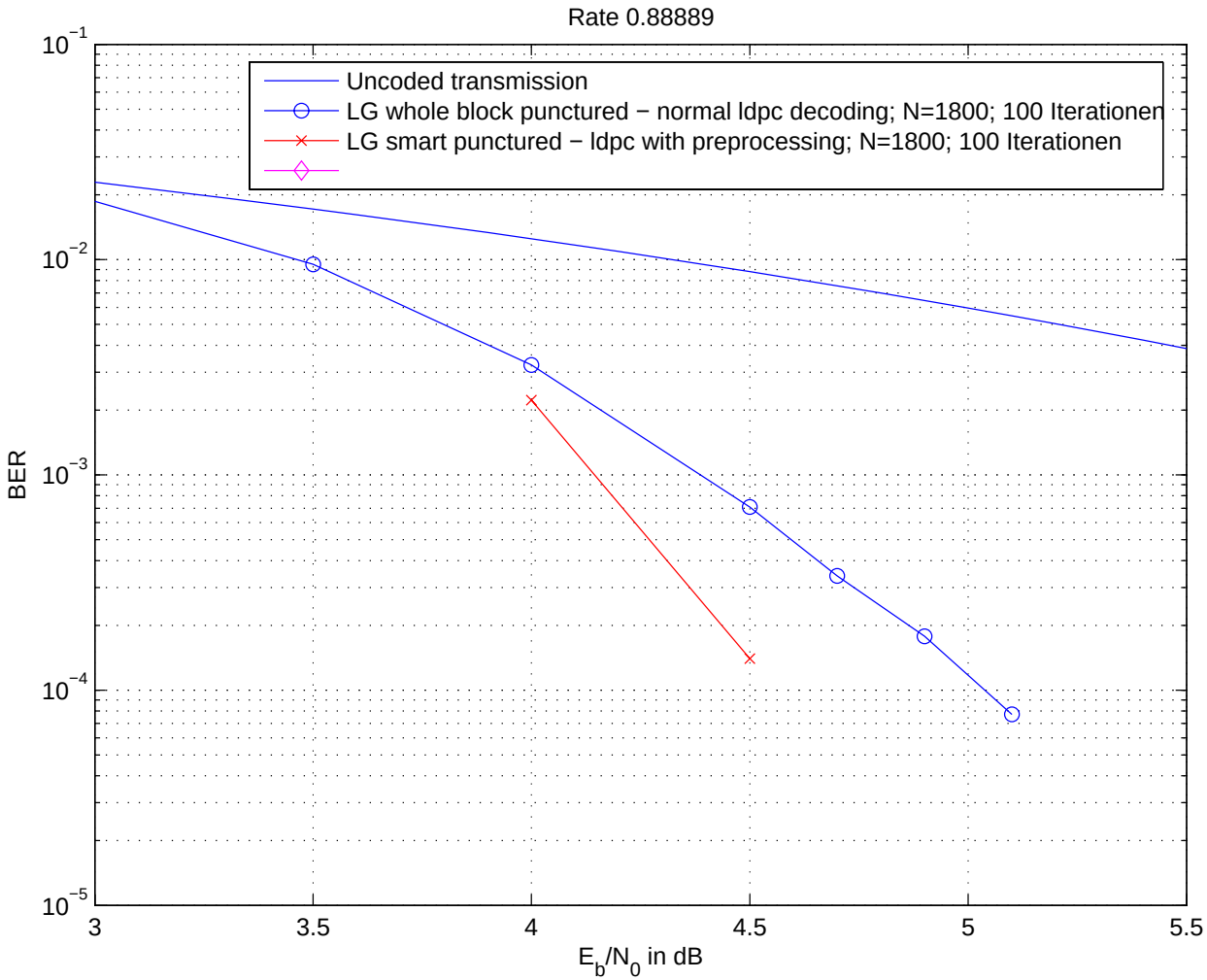
Puncture every second Bit from 201 to 400 $\Rightarrow$ 100 pBits



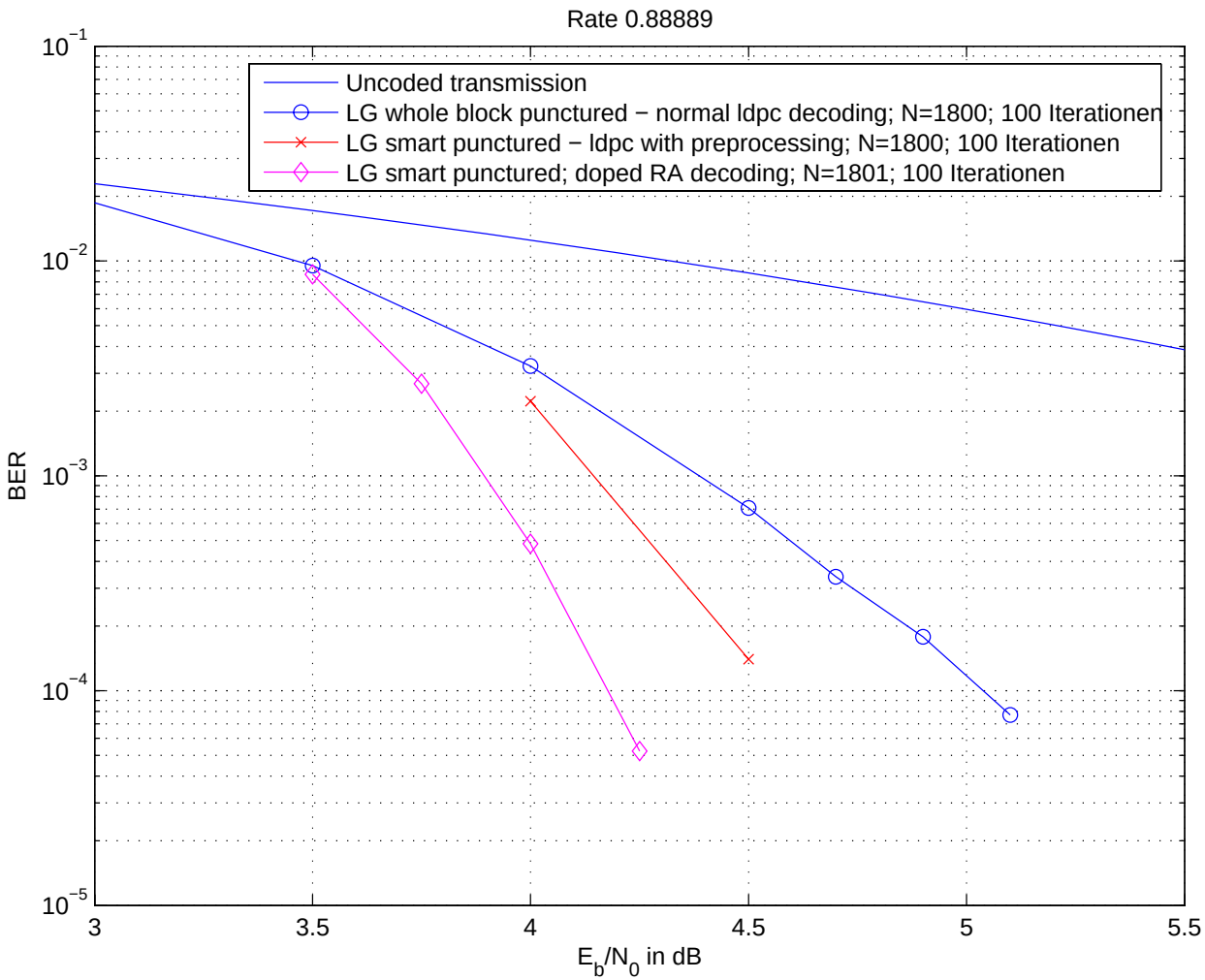
More encouraging results



More encouraging results



More encouraging results



Conclusion

- 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.