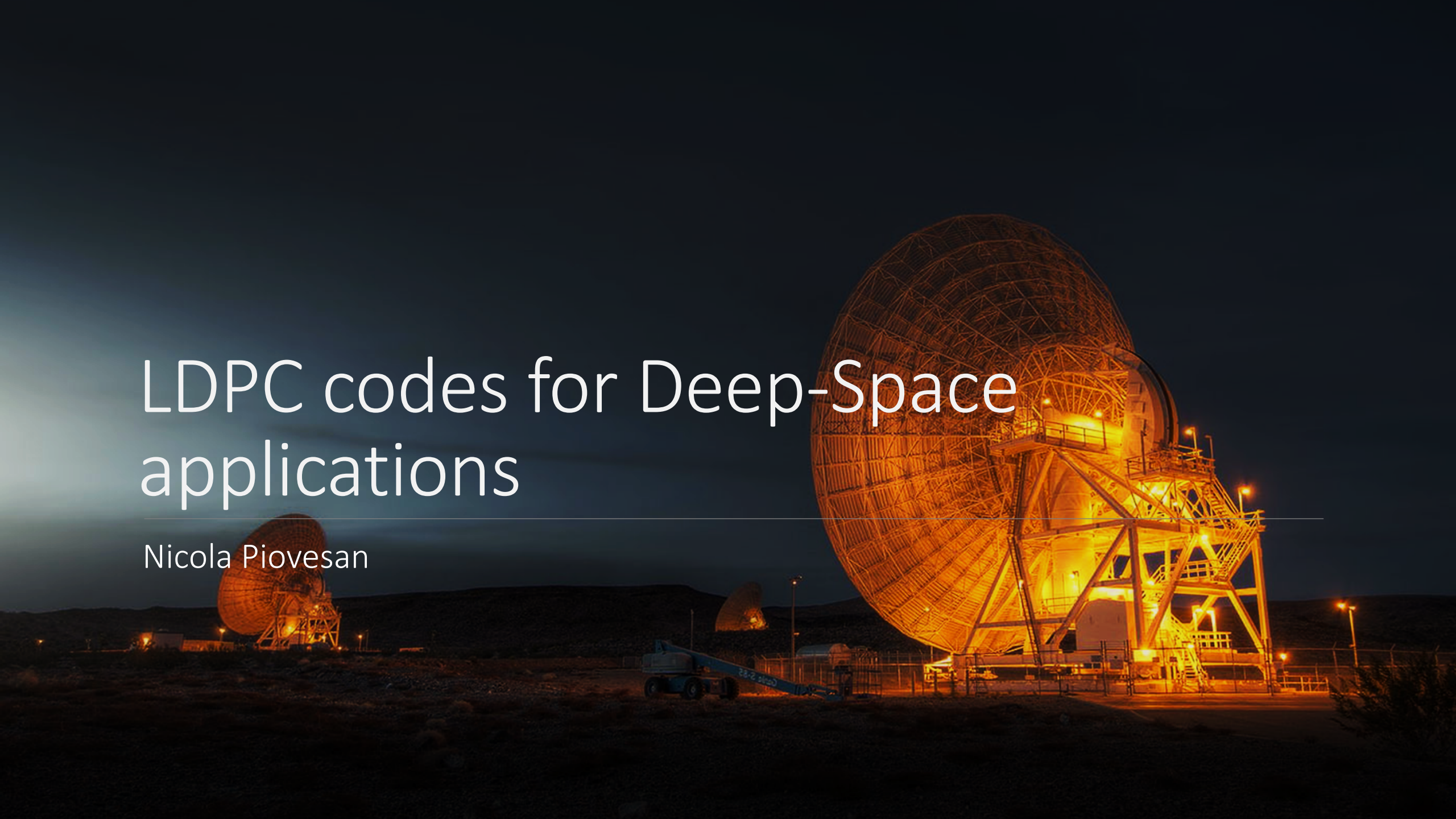


LDPC codes for Deep-Space applications

Nicola Piovesan

Table of Contents

1. Parity check codes
2. LDPC codes
3. Encoder
4. Field of applications;
5. Message passing decoder
6. MatLab/C implementation;
7. Performance evaluation in AWGN settings (BER);

Parity Check Codes

- Parity Check matrix H, used to check if the a codeword is correct or not. It is correct if $H\boldsymbol{y}^T = \mathbf{0}$.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- H is a $m \times n$ binary matrix. Each row corresponds to a parity-check equation and each column of H corresponds to a bit in the codeword.
- To encode a sequence of bit $\boldsymbol{u}$, we need a Generator Matrix, G. The obtained codeword will be $\boldsymbol{c} = \boldsymbol{u}G$.
- The vector $\boldsymbol{u}$ hold the k message bits. For a binary code with k message bits and length n codewords, G is $k \times n$.
- Rate of the code: $^k/_n$.

Parity Check Codes

- Generator matrix for a code with parity-check matrix H can be found by performing **Gauss-Jordan eliminations** on H to obtain

$$H = [A, I_{n-k}]$$

A: $(n-k)\times k$ binary matrix
I_{n-k}: identity matrix of order $n-k$

- The generator matrix is then $G = [I_k, A^T]$

- $GH^T = 0$, holds.

Parity Check Codes

- $d_{\min}$: smallest Hamming distance between any pair of codewords in the code.
- A code with minimum distance $d_{\min}$ can always **detect** t errors, $t < d_{\min}$
- For a code with minimum distance $d_{\min}$, e bit flips **can always be corrected** by choosing the closest codeword (ML) whenever $e \leq \lfloor (d_{\min} - 1)/2 \rfloor$
- The smaller the code rate the smaller the subset of 2^n binary vectors which are codewords and so the **better the minimum distance** that can be achieved by a code with length n.
- An exhaustive search of the closest codeword (ML) is **feasible only for small $\boldsymbol{k}$**. We need to compare each codeword with every one of the 2^k codewords in the code.

Low Density Parity Check Codes

- LDPC codes are block codes with parity check matrices that contain only a **very small number of non-zero entries**. The sparseness of H guarantees both **decoding complexity** which increases linearly with the code length and the **minimum distance** which also increases linearly with the code length.
- Classic block codes are generally decoded with ML like decoding algorithms. LDPC codes are decoded **iteratively** using a graphical representation of their parity-check matrix (Tanner graph).
- LDPC code
  - Regular: if each code bit is contained in a fixed number (d_λ) of parity checks and each parity check equation contains a fixed number (d_ρ) of code bits.
  - Irregular, although.
  - Rate: $R = 1 - \frac{d_\lambda}{d_\rho}$

Encoder

The code which will be considered is the **(8176,7154) code** described in Chapter 7 of the «CCSDS Recommended Standards for TM synchronization and channel coding».

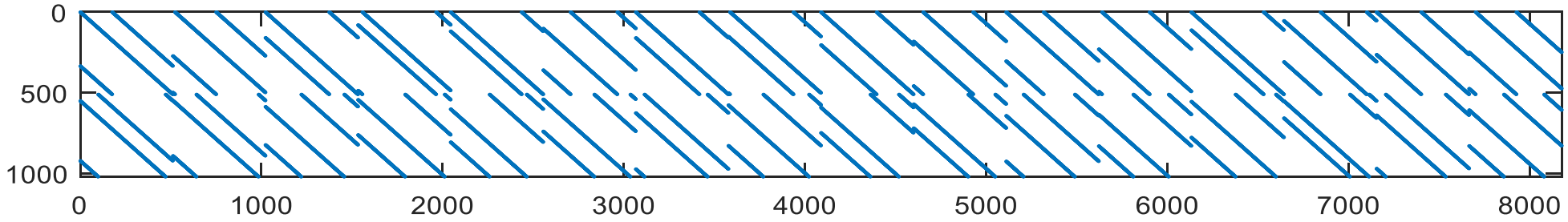
All the performance evaluations will be performed on this code.

The specifications also describes a (8160,7136) code which has the advantage of having all the parameters that are multiples of 32. This is an optimal situation cause current spacecraft and ground systems manipulate and process data at **32 bit computer word size**.

Basic (8176, 7156) LDPC code

The **parity check matrix** is formed by using a 2x16 array of 511x511 square circulants.

⇒This creates a parity check matrix of size 1022x8176 and rank 1020.



Each Ai,j is a 511x511 circulant. **The row weight** of each the 32 circulants is 2. So each row of the parity check matrix has weight 16x2=32.

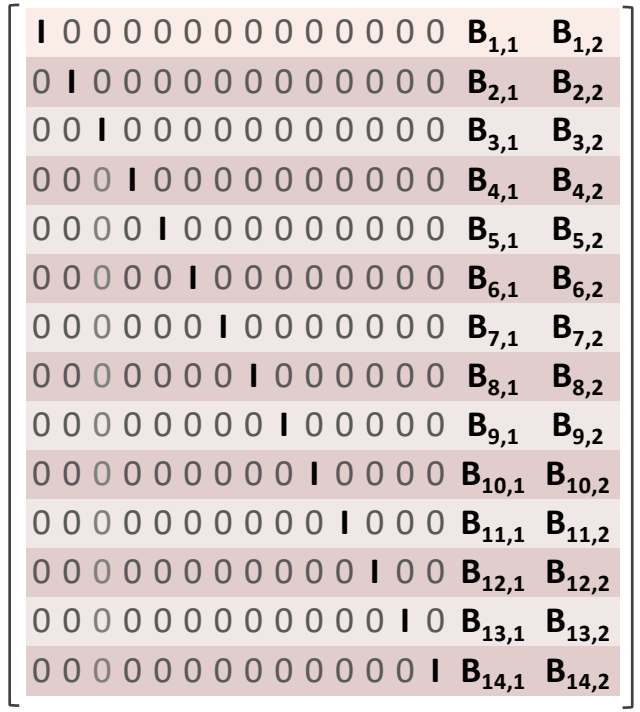
The position of the «1» in the first row of each circulant is defined by a table in the code specification (Table 7-1) while each subsequent row is given by a one-bit right **cyclic shift** of the preceding row.

Encoding Basic (8176,7154) LDPC code

Two bits of information lie outside the structure of the quasi-cyclic encoder and **increase complexity** of the generation matrix. They are not included in the specification so we work with a generation matrix for a systematic **(8176,7154) LDPC subcode** that **can be constructed entirely of circulants**.

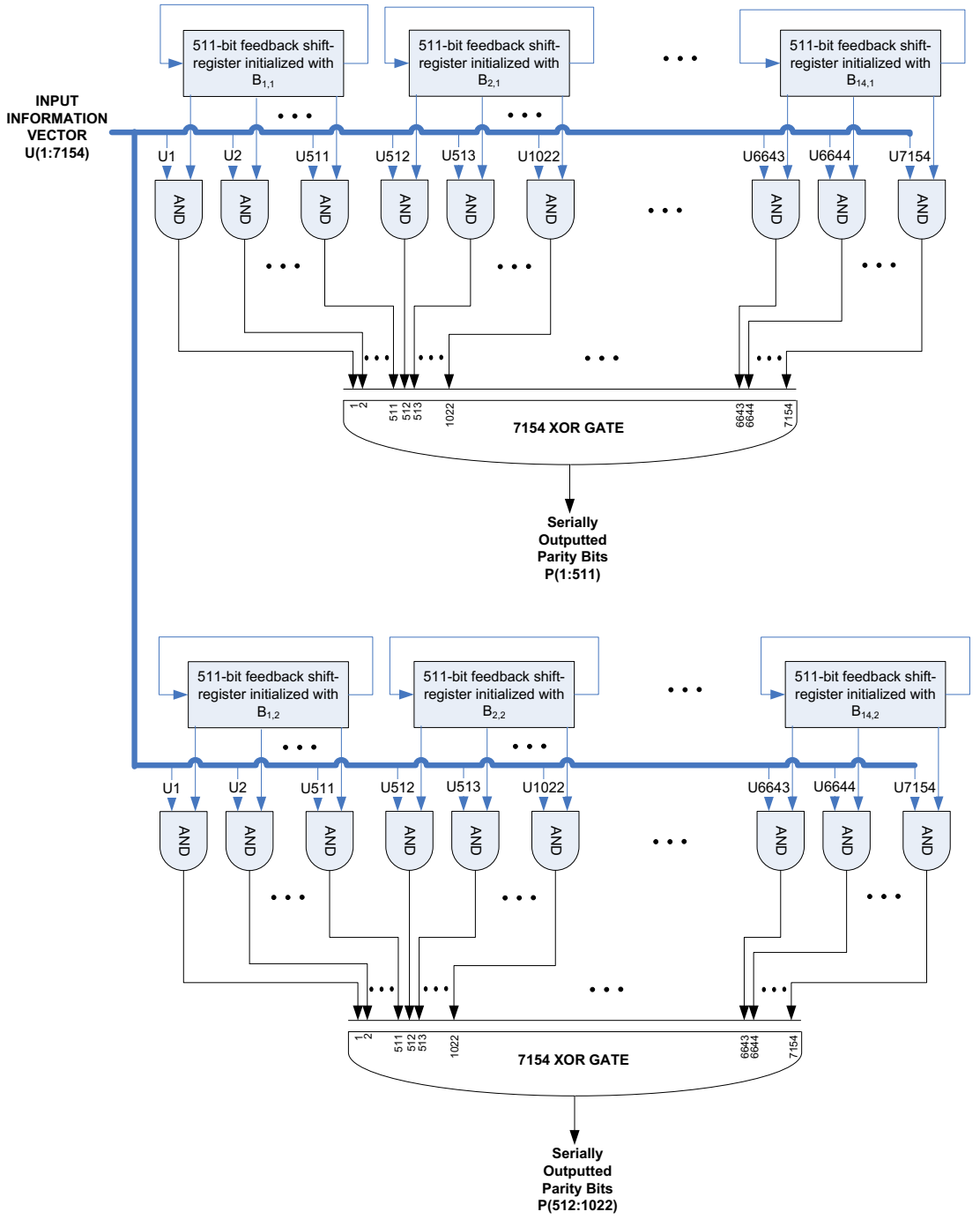
The generation matrix is build as in figure. Each block in figure is a 511x511 circulant so the generation matrix is a 7154x8176 matrix.

Values of $b_{i,j}$s are numerically tabulated in annex C of the code specifications book.



Encoder

Figure: Encoder diagram

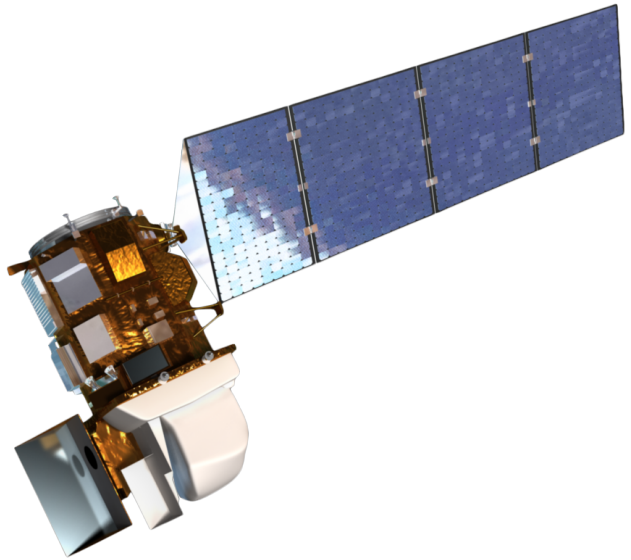


Field of application

In the past, many missions use the concatenated Reed-Solomon and convolutional coding technique for space to ground links. While this standard has served space agencies well in the past, it is **bandwidth inefficient**. The need for bandwidth efficiency has prompted the Microwave and Communication Systems Branch to search for a new channel code that require **less bandwidth without paying a heavy penalty in power requirement and complexity**.

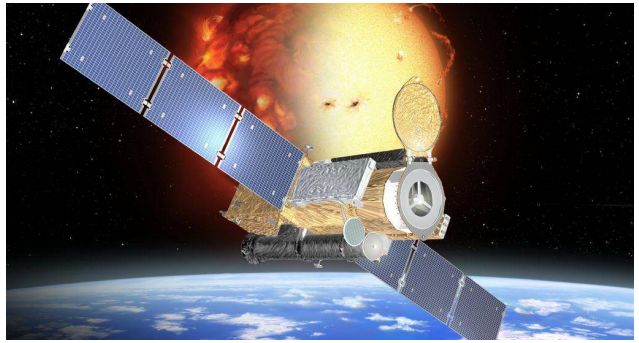
This code is designed for **any space communications link** although it was initially targeted for communications links between spacecraft to ground elements. Current advances in space qualified field programmable gate array (FPGA) technology has potentially given rise to implementations for space to space communication links as well. As a result, this code can apply to **any space link that requires bandwidth efficient communications**.

Applications



Landsat Data Continuity Mission
Launched: February, 2013
Earth-observing sensors.
This code is used to downlink data transmission in the X-band. This 384 Mbps data rate communications link is NASA's first operational use of an LDPC and is the first use of an LDPC code for a space to ground link for any agency or company.

IRIS
Launched: June, 2013
Solar observations



GOES-R
Planned: 2016
Weather satellite

Message passing decoder

The decoding is done using a **message-passing algorithm**. This is an iterative decoding technique as the messages pass back and forward between the bit and check nodes iteratively until a result is achieved (or the process halted).

The sent messages are **probabilities** which represent a level of belief about the value of the codeword bits.

It is convenient to represent probability values as **log likelihood ratios**. This kind of decoding is called **sum-product decoding** since the use of LLRs allows the calculations at the bit and check nodes to be computed using sum and product operations.

Sum-product decoding

It is a soft decision message-passing algorithm which accepts the probability of each received bit as input.
We use **LLRs to reduce the algorithm complexity**, which can be computed as

$$L(x) = \log\left(\frac{p(x=0)}{p(x=1)}\right)$$

So, if $p(x = 0) > p(x = 1)$ then $L(x)$ is positive and vice versa. The sign of $L(x)$ provides the hard decision on x while the magnitude $|L(x)|$ is the reliability of this decision.

Benefit: when probabilities need to be multiplied, LLRs need only to be added! → **complexity reduction**.

The aim of this algorithm is to compute the maximum a posteriori probability (MAP) for each codeword bit.

Initialization

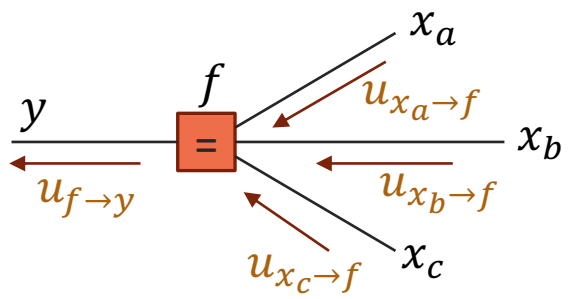
Considering we are using AWGN channel, we obtain the following result

$$\mathrm{LLR}_{g_l \to c_l} = \log \frac{g_l(0)}{g_l(1)} = \log \frac{\frac{1}{\sqrt{2\pi\sigma_w^2}} e^{-\frac{(r_l+1)^2}{2\sigma_w^2}}}{\frac{1}{\sqrt{2\pi\sigma_w^2}} e^{-\frac{(r_l-1)^2}{2\sigma_w^2}}} = -2\frac{r_l}{\sigma_w^2}$$

$$\sigma_w = \sqrt{4/7 \cdot 10^{-\left(\frac{E_b}{N_0}\right)_{dB}/10}}$$

Variable nodes

$$f(y, x_a, x_b, x_c) = \delta_{y,x_a}\delta_{y,x_b}\delta_{y,x_c}$$



$$u_{f\to y}(y) = \sum_{x_a,x_b,x_c} f(y, x_a, x_b, x_c)\, u_{x_a\to f}(x_a) u_{x_b\to f}(x_b) u_{x_c\to f}(x_c) = u_{x_a\to f}(y) u_{x_b\to f}(y) u_{x_c\to f}(y)$$

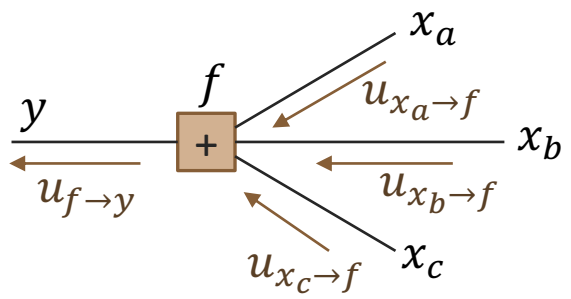
$$\mathrm{LLR}_{f\to y} = \log\left(\frac{u_{f\to y}(0)}{u_{f\to y}(1)}\right) = \log\left(\frac{u_{x_a\to y}(0) u_{x_b\to y}(0) u_{x_c\to y}(0)}{u_{x_a\to y}(1) u_{x_b\to y}(1) u_{x_c\to y}(1)}\right) = \sum_i \mathrm{LLR}_{x_i\to f}$$

Check nodes

$$f(y, x_a, x_b, x_c) = \delta_{y, x_a + x_b + x_c}$$

$$u_{f \to y}(y) = \sum_{x_a, x_b, x_c} f(y, x_a, x_b, x_c) u_{x_a \to f}(x_a) u_{x_b \to f}(x_b) u_{x_c \to f}(x_c)$$

$$= \sum_{x_a, x_b, x_c} u_{x_a \to f}(y + x_b + x_c)\, u_{x_b \to f}(x_b) u_{x_c \to f}(x_c)$$



$$\mathrm{LLR}_{f \to y} = \log\left(\frac{u_{f \to y}(0)}{u_{f \to y}(1)}\right) = \log\left(\frac{\sum_{x_a, x_b, x_c} \delta_{0, x_a + x_b + x_c} u_{x_a \to f}(x_a)\, u_{x_b \to f}(x_b) u_{x_c \to f}(x_c)}{\sum_{x_a, x_b, x_c} \delta_{1, x_a + x_b + x_c} u_{x_a \to f}(x_a)\, u_{x_b \to f}(x_b) u_{x_c \to f}(x_c)}\right)$$

Even # «1»

Odd # «1»

Check nodes

$$\mathrm{LLR}_{f\to y} = \log\left(\frac{u_{f\to y}(0)}{u_{f\to y}(1)}\right) = \log\left(\frac{\sum_{x_a,x_b,x_c}\delta_{0,x_a+x_b+x_c}u_{x_a\to f}(x_a)\,u_{x_b\to f}(x_b)u_{x_c\to f}(x_c)}{\sum_{x_a,x_b,x_c}\delta_{1,x_a+x_b+x_c}u_{x_a\to f}(x_a)\,u_{x_b\to f}(x_b)u_{x_c\to f}(x_c)}\right)$$

Let's consider the 2-input case

$$\mathrm{LLR}_{f\to y} = \log\left(\frac{u_{f\to y}(0)}{u_{f\to y}(1)}\right) = \log\left(\frac{u_{x_1\to f}(1)u_{x_2\to f}(1) + u_{x_1\to f}(0)u_{x_2\to f}(0)}{u_{x_1\to f}(0)u_{x_2\to f}(1) + u_{x_1\to f}(1)u_{x_2\to f}(0)}\right)$$

$$= \log\left(\frac{1+\frac{u_{x_1\to f}(0)u_{x_2\to f}(0)}{u_{x_1\to f}(1)u_{x_2\to f}(1)}}{\frac{u_{x_1\to f}(0)}{u_{x_2\to f}(1)}+\frac{u_{x_2\to f}(0)}{u_{x_2\to f}(1)}}\right) = \log\left(\frac{1+e^{\mathrm{LLR}_{x_1\to f}+\mathrm{LLR}_{x_2\to f}}}{e^{\mathrm{LLR}_{x_1\to f}}+e^{\mathrm{LLR}_{x_2\to f}}}\right)$$

Check nodes

Let's define the $\emptyset$ function

$$\emptyset(x) = \frac{e^x - 1}{e^x + 1} = \tanh\frac{x}{2}$$

$$\emptyset^{-1}(x) = \log\left(\frac{1+x}{1-x}\right)$$

Remember

$$\mathrm{LLR}_{f\to y} = \log\left(\frac{1 + e^{\mathrm{LLR}_{x_1\to f}+\mathrm{LLR}_{x_2\to f}}}{e^{\mathrm{LLR}_{x_1\to f}} + e^{\mathrm{LLR}_{x_2\to f}}}\right)$$

$$\emptyset\left(\mathrm{LLR}_{f\to y}\right) = \frac{\frac{1 + e^{\mathrm{LLR}_{x_1\to f}+\mathrm{LLR}_{x_2\to f}}}{e^{\mathrm{LLR}_{x_1\to f}} + e^{\mathrm{LLR}_{x_2\to f}}} - 1}{\frac{1 + e^{\mathrm{LLR}_{x_1\to f}+\mathrm{LLR}_{x_2\to f}}}{e^{\mathrm{LLR}_{x_1\to f}} + e^{\mathrm{LLR}_{x_2\to f}}} + 1} = \frac{\left(e^{\mathrm{LLR}_1} - 1\right)\left(e^{\mathrm{LLR}_2} - 1\right)}{\left(e^{\mathrm{LLR}_1} + 1\right)\left(e^{\mathrm{LLR}_2} + 1\right)} = \emptyset(\mathrm{LLR}_1)\emptyset(\mathrm{LLR}_2)$$

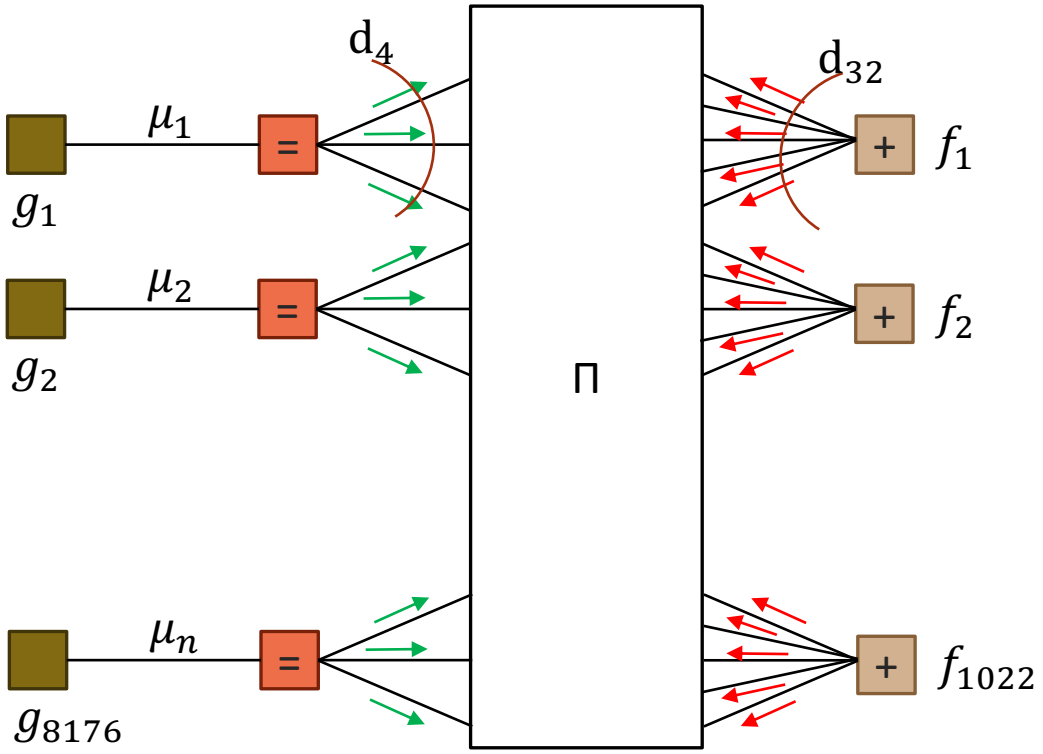
$$\Longrightarrow \emptyset\left(\mathrm{LLR}_{f\to y}\right) = \prod_i \emptyset(\mathrm{LLR}_i)$$

Check nodes

Finally we obtain the following formula for the check nodes update

$$LLR_{f\to y} = \emptyset^{-1}\left(\prod_i \emptyset(LLR_i)\right) = \log\frac{1+\prod_i \tanh\left(\frac{LLR_i}{2}\right)}{1-\prod_i \tanh\left(\frac{LLR_i}{2}\right)}$$

Scheme

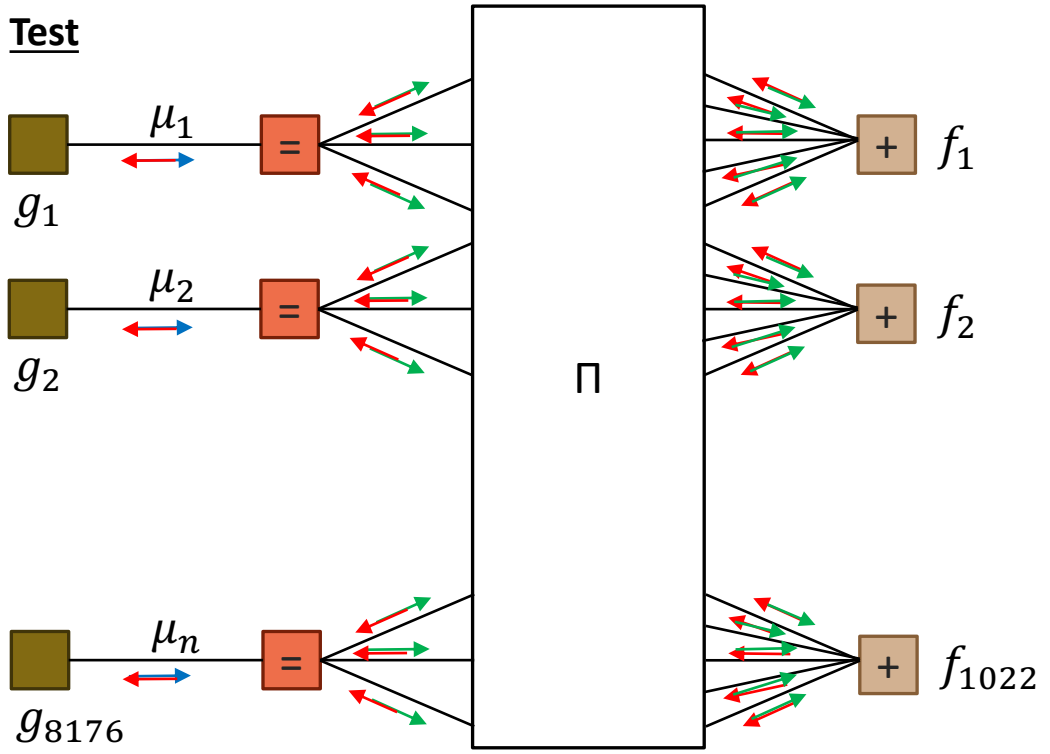


Scheme

Initialization:

$$\mathrm{LLR}_{g_l \to c_l} = \log -2\frac{r_l}{\sigma_w^2}$$

Test



Check messages:

$$LLR_{f \to y} = \log \frac{1 + \prod_i \tanh\left(\frac{LLR_i}{2}\right)}{1 - \prod_i \tanh\left(\frac{LLR_i}{2}\right)}$$

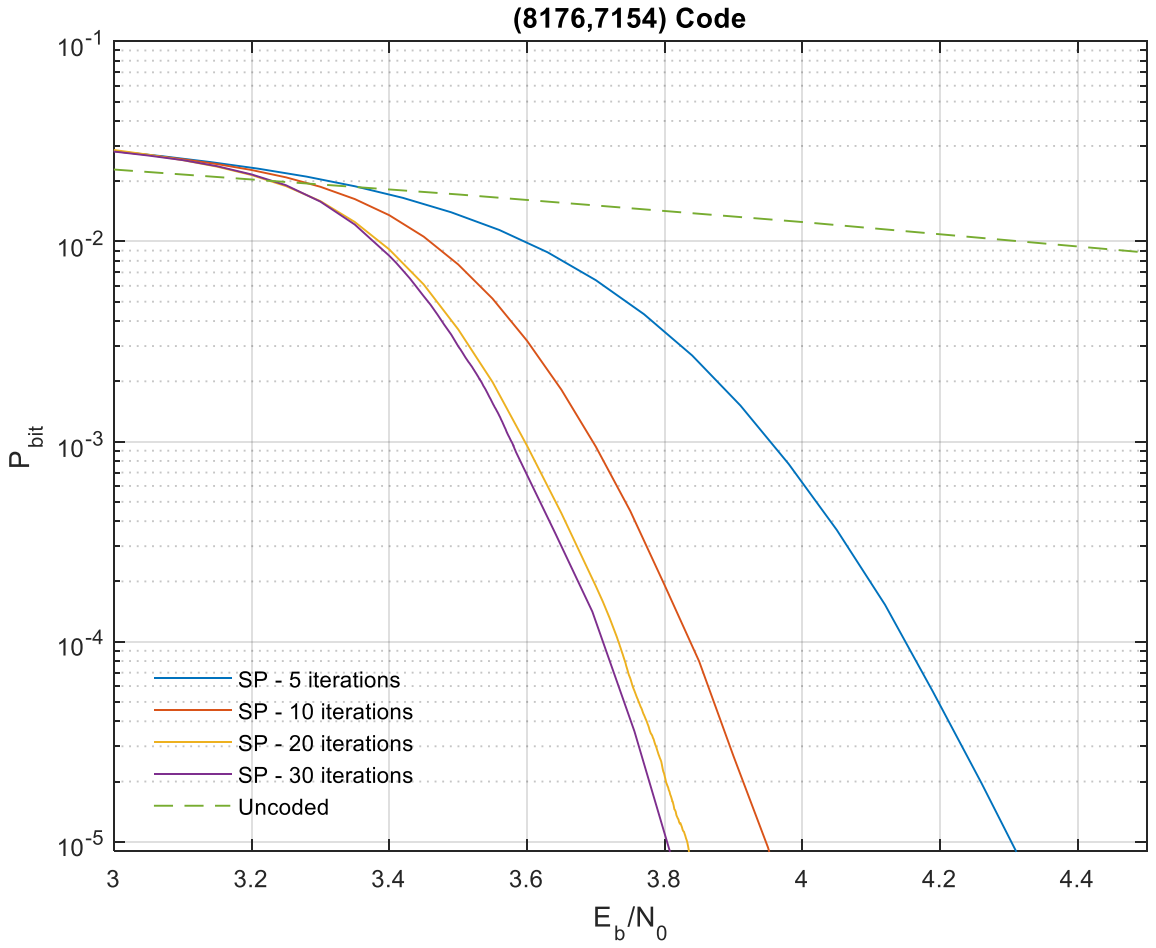
Bit messages:

$$\mathrm{LLR}_{f \to y} = \sum_i \mathrm{LLR}_{x_i \to f}$$

Matlab/C implementation

- Randomly generate a bit sequence of length 7154;
- Encode the sequence -> We obtain a codeword of length 8176;
- Map to -1, +1;
- Add noise (considering the E_b/N_0 ratio we are testing);
- Messages from bit-nodes are stored in matrix M
  M(c,b) <- Store message from bit-node b to check-node c
- Messages from check-nodes are stored in matrix E
  E(c,b) <- Store message from check-node c to bit-node b
- At each step, test the decoded word using matrix H.

Results – Sum-product algorithm

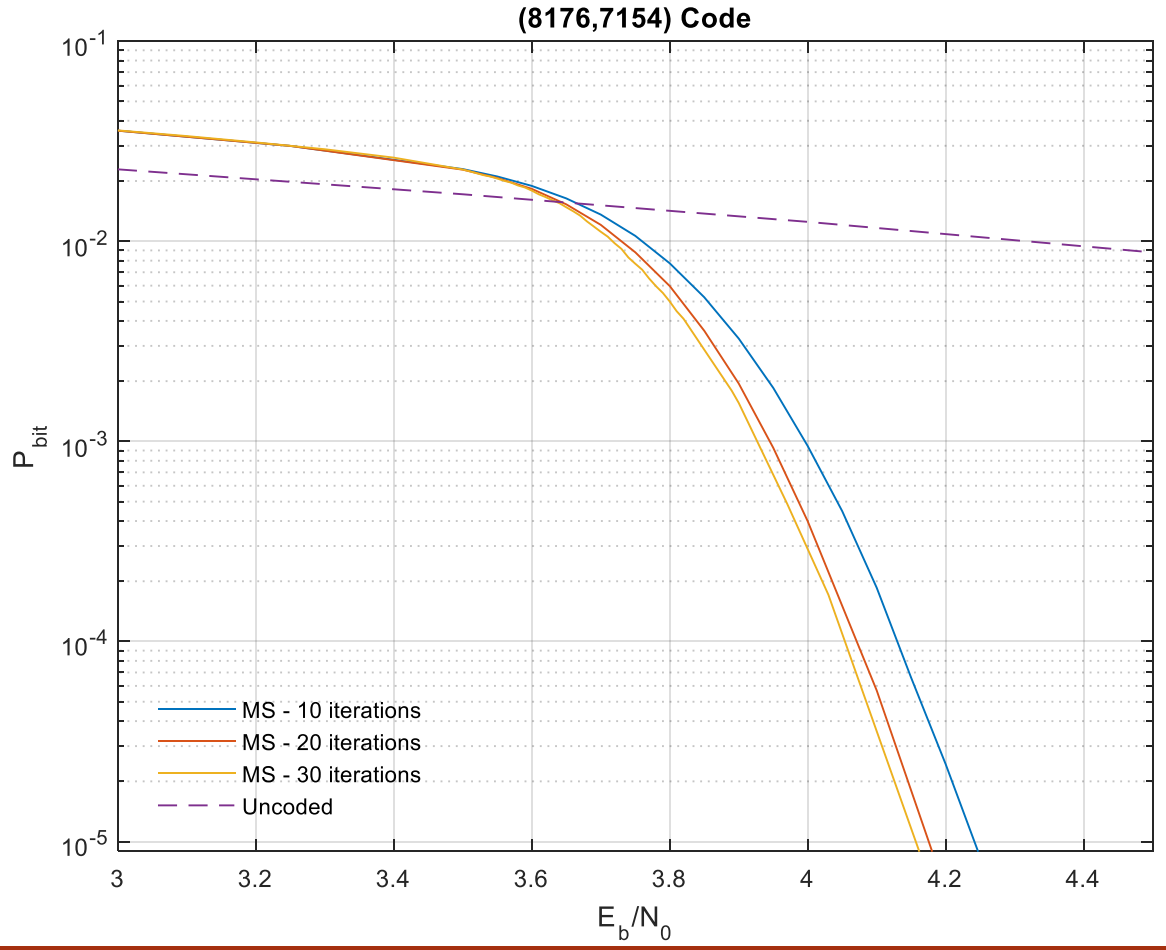


Min-sum variation

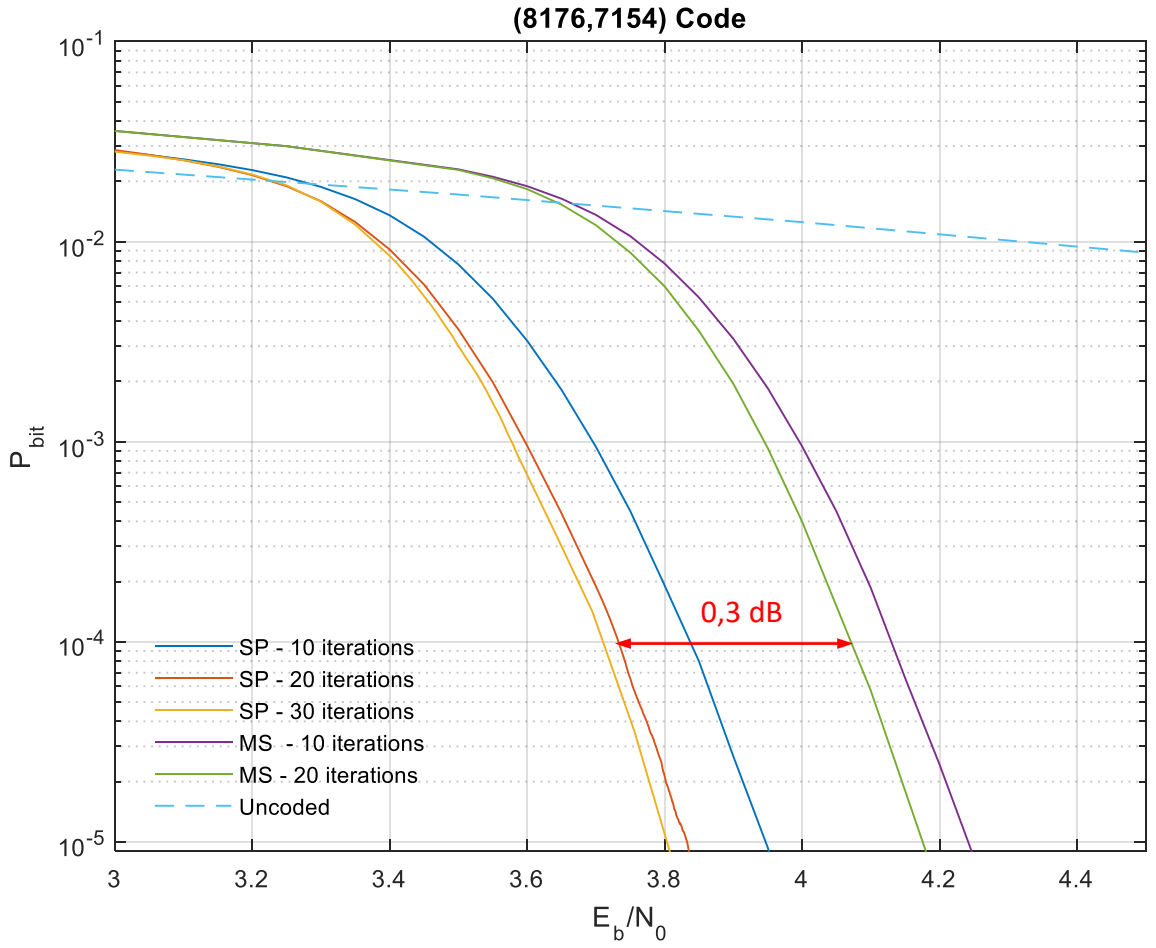
Check nodes computation can be optimized in terms of complexity, considering than in the output message computation, **the term corresponding to the smallest LLR, dominates the product term**, so the product can be **approximated by a minimum**.

$$LLR_{f\to y} \approx \left(\prod_i \operatorname{sign}(\mathrm{LLR}_i)\right) \min_i |\mathrm{LLR}_i|$$

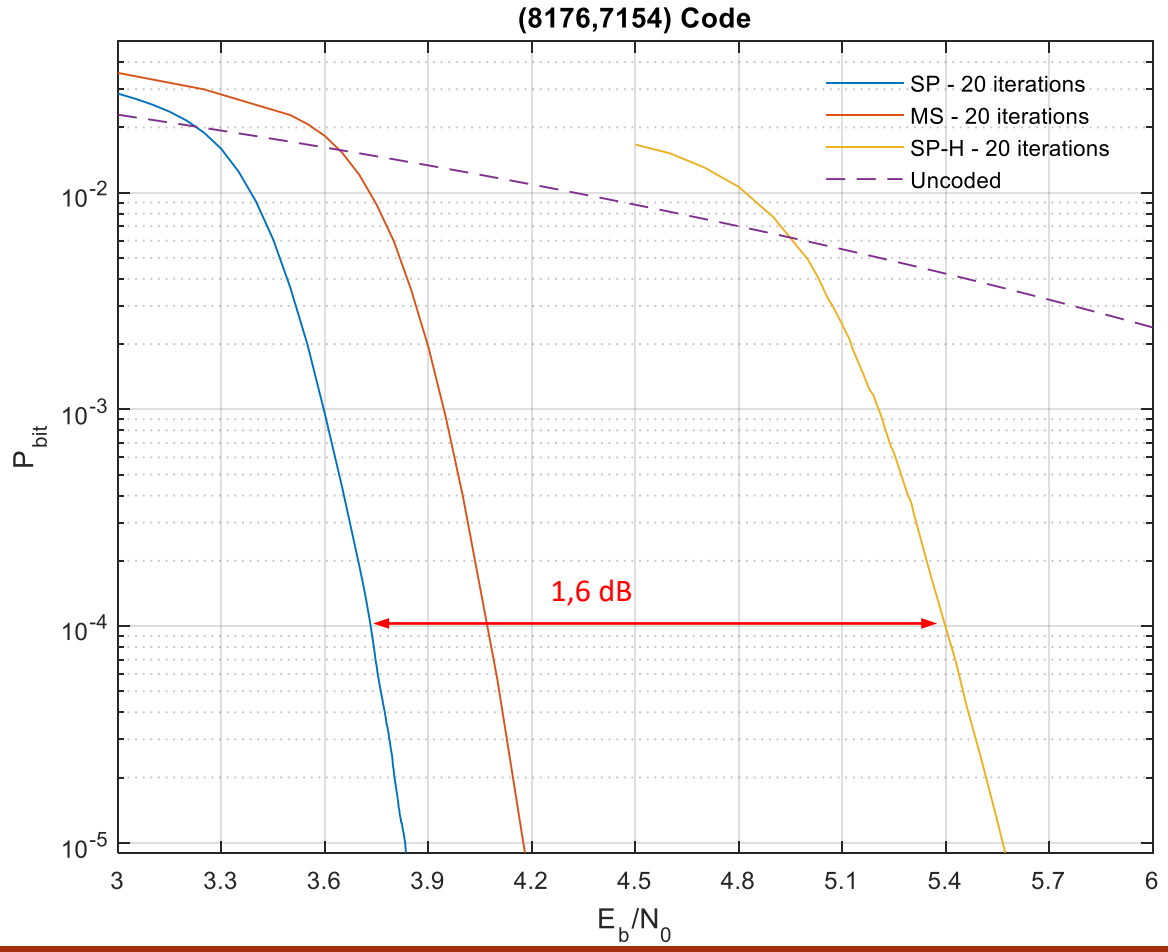
Result – Min-sum algorithm



Results – Sum-product vs. Min-sum



Result – Hard decoding



End