Implementation and Performance Evaluation of ITU-T G.975.1 LDPC Binary Code

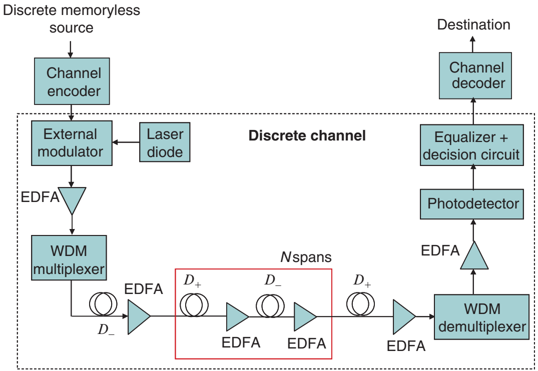
Channel Coding 15/16 Final Project

Michele Polese

2016-03-11

Outline

- Field of application: DWDM submarine systems
- LDPC Encoder: encoding matrix and implementation
- Message Passing Decoder
- LDPC Decoder `C++` Implementation: the flexibility of OOP
- Performance Evaluation
- Conclusions



- DWDM interfaces with different optical transport networks,
- The channel can be modeled as a Gaussian channel.

ITU-T G.975.1

Forward error correction for high bit-rate DWDM submarine systems

- *Super FEC* schemes for coding in submarine optical systems,
- More robust than ITU-T G.975 FEC - RS (255, 239),
- Concatenate RS or BCH with different options,
- **Low Density Parity Check** code LDPC (32640, 30592).

- Information word with $K = 30592$, it fits a RS (255,239) frame,
- High coding rate $R = \frac{K}{N} = 0.9374$,
- Spectral efficiency $\rho = \dfrac{R \log_2(M)}{BT} = 2R = 1.8748$,
- Hardware implementation suitable for application with 10G and 40G fibers.

- The information bit are placed in a 112×293 matrix $\mathbf{S}$,
- Bit j, $j \in [1, 30592]$, is inserted in position $(r, 293r + 292 - q)$ with

$$r = \left\lfloor \frac{j}{293} \right\rfloor$$

$$q = j + 172$$

- Entries in $(0, 292 - d), d \in [0, 172]$ are set to 0 and never transmitted,
- 7 slopes s_i, $i \in \{1 \ldots 7\}$, are chosen,
- For each slope s_i 293 lines are defined by

$$(a, b) | b = (s_i a + c)_{\bmod 293}, \quad c \in [0, 292]$$

- $293 \times 7 = 2051$ lines are defined,
- The sum (modulo 2) of the bits in each line must be 0,
- The parity check equations define a system of 2051 equations in 2051 unknowns,
- 6 parity check bit are redundant, and removed from the linear system, as well as the last equation ($c = 292$) for the first 6 slopes,
- This system can be written as

$$\mathbf{Hc} = \mathbf{0}$$

Matrix $\mathbf{H}$

$$\mathbf{H} = \begin{array}{c c} & \begin{array}{cc} 105\times 293 & 7\times 293-6 \end{array} \\ 2045 & \left[\begin{array}{cc} \quad\mathbf{M}\quad & \quad\mathbf{N}\quad \end{array} \right] \end{array}$$

Row i of $\mathbf{H}$ is defined by a valid couple (s_i, c_i), and column j corresponds to bit $(\lfloor j/293 \rfloor, j_{\bmod 293})$ in matrix $\mathbf{S}$. Then

$$h_{i,j} = \begin{cases} 1, & \text{if } j_{\bmod 293} == s_i \left\lfloor \dfrac{j}{293} \right\rfloor + c_i \\ 0, & \text{otherwise} \end{cases}$$

Given the line (s_i, c_i), a column of $\mathbf{M}$ contains a 1 if the *information* bit in the related position belongs to the line, $\mathbf{N}$ if a *parity check* bit belongs to the line.

H is transformed to compute the encoding matrix **G**

$$\mathbf{H}_{toinv} = \begin{array}{c} \\ 2045 \end{array} \begin{array}{c} \begin{array}{cc} 2045 & 30765 \end{array} \\ \left[\begin{array}{cc} \mathbf{N} & \mathbf{M} \end{array} \right] \end{array}$$

$$\mathbf{H}_{toinv} \mid \mathbf{I}_{2045} = \begin{array}{c} \\ 2045 \end{array} \begin{array}{c} \begin{array}{ccc} 2045 & 30765 & 2045 \end{array} \\ \left[\begin{array}{ccc} \mathbf{N} & \mathbf{M} & \mathbf{I}_{2045} \end{array} \right] \end{array}$$

Gauss elimination is applied to bring $\mathbf{H}_{toinv} \mid \mathbf{I}_{2045}$ in a row echelon form, then Jordan algorithm is used to isolate an identity matrix in first 2045 columns. The result is

$$\begin{array}{c} \\ 2045 \end{array} \begin{array}{c} \begin{array}{ccc} 2045 & 30765 & 2045 \end{array} \\ \left[\begin{array}{ccc} \mathbf{I}_{2045} & \mathbf{N}^{-1}\mathbf{M} & \mathbf{N}^{-1} \end{array} \right] \end{array}$$

and finally

$$\tilde{\mathbf{H}} = \begin{array}{c} \\ 2045 \end{array} \begin{array}{c} \begin{array}{cc} 30765 & 2045 \end{array} \\ \left[\begin{array}{cc} \mathbf{N}^{-1}\mathbf{M} & \mathbf{I}_{2045} \end{array} \right] \end{array}$$

Matrix **G** is obtained as

$$\mathbf{G} = \begin{array}{c c} & 30765 \\ 30765 & \left[\begin{array}{c} \mathbf{I}_{30765} \\ \mathbf{N}^{-1}\mathbf{M} \end{array}\right. \\ 2045 & \end{array}\left.\begin{array}{c} \\ \\ \\ \end{array}\right]$$

For the Gauss elimination NTL[1] library is used. Then each row of matrix $\mathbf{K} = \mathbf{N}^{-1}\mathbf{M}$ is saved into a `std::bitset` and stored to file.

[1] `http://www.shoup.net/ntl/`

- The encoder is implemented as a `C++` object. Upon initialization, matrix $\mathbf{K}$ is read from file and loaded in memory,
- Both infoword and codeword are `std::bitset`,
- The first 30592 bit of the codeword are filled with the information word, then 2045 parity check bit are computed with an `and` operation between the infoword and the corresponding row of matrix $\mathbf{K}$,
- Three zero bit are inserted between the information word and the parity check bits.

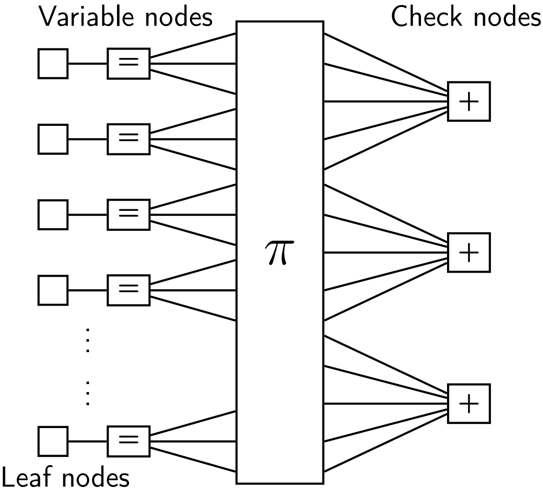


Figure: Factor graph for LDPC decoding

- The decoder is based on this factor graph,
- Decoding is performed in the LLR domain.

The LDPC code under analysis is a binary code. Therefore the LLR associated to message μ is expressed as

$$LLR_{\mu} = \ln\left(\frac{\mu(0)}{\mu(1)}\right)$$

Leaf nodes are initialized with received values, and under the hypothesis of equally probable input symbols the LLR is

$$LLR_{g_l \to =_l} = \ln\left(\frac{\frac{1}{\sqrt{2\pi\sigma_w^2}} e^{-\frac{1}{2\sigma_w^2}(r_l+1)}}{\frac{1}{\sqrt{2\pi\sigma_w^2}} e^{-\frac{1}{2\sigma_w^2}(r_l-1)}}\right) = -\frac{2r_l}{\sigma_w^2}$$

A **variable node** represents a delta function, therefore the LLR on each branch is

$$LLR_{=\rightarrow j} = \sum_{i \neq j} LLR_{i \rightarrow =}$$

This LDPC code has variable nodes with 7 branches connected to check nodes, with the exception of variables figuring in linearly dependent parity check equations, which have 6 outgoing branches.

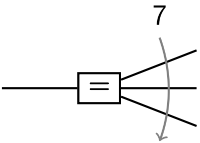


Figure: Variable Node

Each **check node** is connected to 112 variable nodes, and there are 2045 check nodes. With Sum Product algorithm, the LLR of outgoing branch j is given by

$$LLR_{+\rightarrow j} = \tilde{\Phi}\left(\sum_{i\neq j}\tilde{\Phi}\left(|LLR_{i\rightarrow +}|\right)\right)\Pi_{i\neq j}\text{sign}\left(LLR_{i\rightarrow +}\right)$$

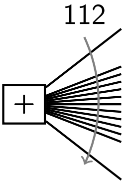
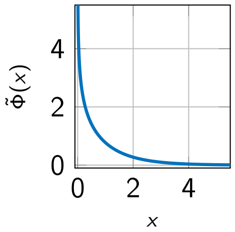


Figure: Check Node

- The function $\tilde{\Phi}(x)$ is given by $\tilde{\Phi}(x) = -\ln\left(\tanh\left(\frac{1}{2}x\right)\right)$



- Min Sum algorithm was implemented too, by changing the update function in the check node

$$LLR_{+\to j} = \min_{i\neq j}\left\{\left|LLR_{i\to +}\right|\right\} \Pi_{i\neq j}\text{sign}\left(LLR_{i\to +}\right)$$

The **marginalization** is carried out between leaf nodes g_l and variable nodes $=_l$, thus

$$\hat{x} = \begin{cases} 0, & \text{if } LLR_{g_l \to x} + LLR_{=_l \to x} \geq 0 \\ 1, & \text{otherwise} \end{cases}$$

LDPC codes have cycles. Therefore to decode we need

- **Initialization**: Variable nodes are initialized with the leaf node LLR
- **Schedule**:
  1. Run message passing on check nodes $+$ and update their outgoing LLRs
  2. Run message passing on variable nodes $=$
  3. Marginalize: **if** a codeword is found or if the maximum number of attempts is reached *stop*, **else** go to 1

The message passing decoder was implemented using the **flexibility** offered by *Object Oriented Programming* (OOP).

- `VariableNode` **class** represents a single variable node, and it is initialized with a position in the standard matrix **S** and the index of its 7 check nodes,
- `CheckNode` **class** represents a single check node, it knows to which variable node is connected to,
- `LdpcDecoder` **class** contains a vector of variable nodes, a vector of check nodes and a vector of received LLR. It handles initialization, the update schedule and marginalization.

C++ Implementation 2/3

- `LdpcDecoder` is initialized once per simulation campaign,
- The $\tilde{\Phi}(x)$ function is clipped to `infinity()` for $x < 10^{-300}$ and to 0 for $x > 38$,
- The Sum Product update computes once all the $\tilde{\Phi}$ values, sums them and the subtracts the outgoing $\tilde{\Phi}$,
- **Testing**: update in variable and check node is tested to check if the results obtained are as expected.

- For each simulation, a single noise vector is generated, and scaled by σ_w for each $\frac{E_b}{N_0}$ with

$$\sigma_w = \frac{1}{\sqrt{2\frac{E_b}{N_0}R}}$$

- The decoding for each $\frac{E_b}{N_0}$ is launched in a separate thread, to parallelize computations.

Results: BER for different max number of iterations

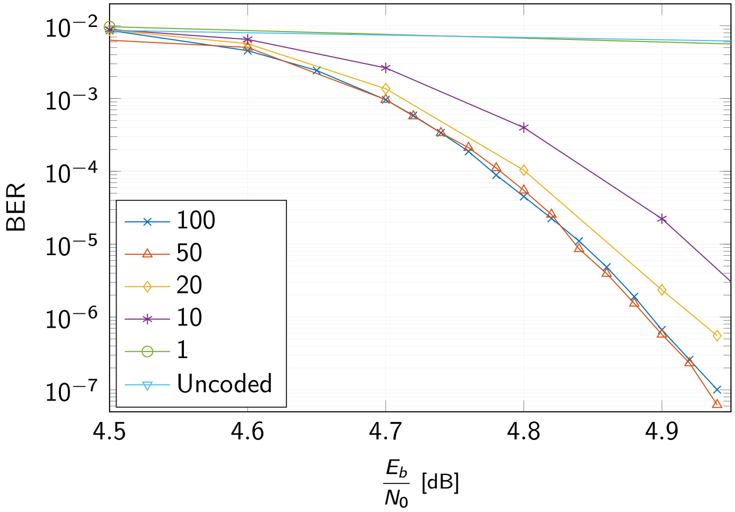


Figure: BER for different number of iterations

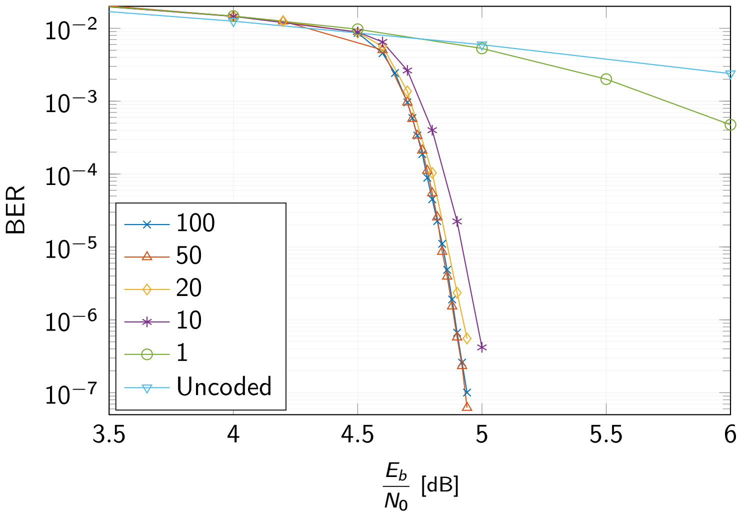


Figure: Waterfall is reached with 50 iterations

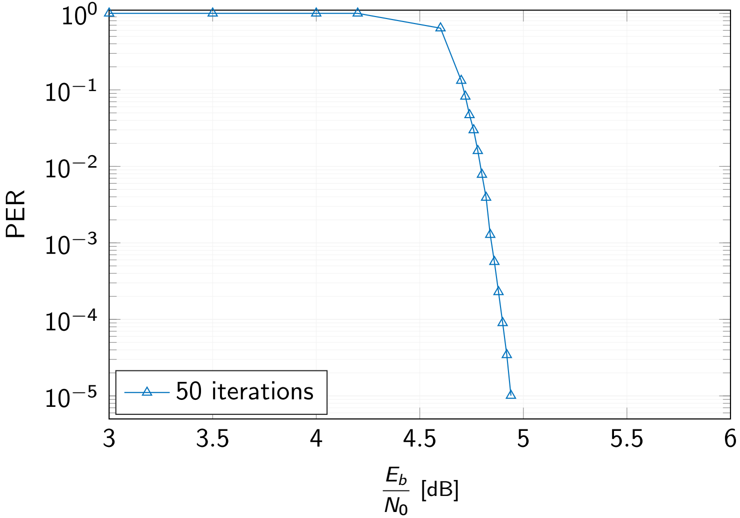


Figure: PER for 50 iterations

Results: comparison with a different choice of slopes

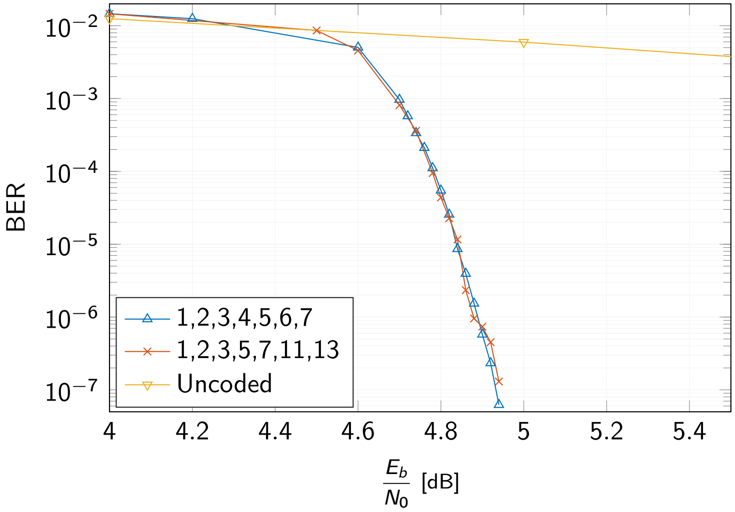


Figure: Comparison with 2 different set of slopes

Results: comparison with Min Sum algorithm

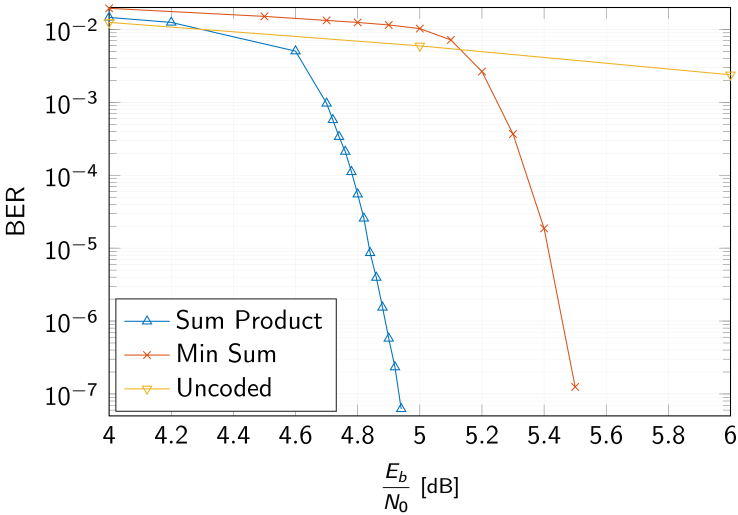


Figure: Sum Product vs Min Sum algorithm, 50 iterations

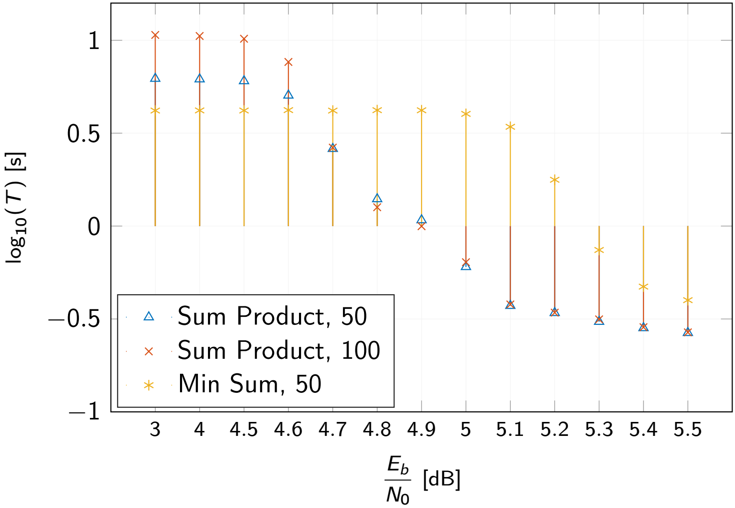


Figure: Average time to decode a packet

Conclusions

- LDPC code for DWDM submarine systems was presented,
- Encoding and message passing decoding were described,
- The `C++` implementation was detailed,
- Results show that Sum Product algorithm exhibits a waterfall between 4.6 and 4.9 dB,
- Min Sum algorithm exhibits a 0.6 dB gap with respect to Sum Product,
- The code is available on Github `https://github.com/mychele/channelcoding1516`.

Implementation and Performance Evaluation of ITU-T G.975.1 LDPC Binary Code

Channel Coding 15/16 Final Project

Michele Polese

2016-03-11