Error-correcting codes

Lecture 9. Low-density parity-check codes. Part 2
Agenda

- Motivation
- Definitions and notations. Construction
- Single-parity check codes
- Belief propagation (BP) decoding
- Tanner graphs

Part 2:
- Quasi-cyclic codes
- Encoding
- Search for good codes
- Practical algorithms
- etc.
Motivation

- LDPC codes are codes with low decoding complexity and high performance
- Tanner graph describes message exchange between variable and check nodes
- The larger girth of Tanner graph the more independent iterations of BP decoding are available
- We still do not know how to construct good codes. We need some structure in order to simplify both encoding and decoding
- We have to solve implementation issues: arithmetic, parallel implementation, etc.
A hypergraph is a generalization of a graph in which the hyperedges are subsets of vertices and may connect (contain) any number of vertices. A hypergraph is called s-uniform if every hyperedge connects s vertices. The degree of a vertex in a hypergraph is the number of hyperedges that are connected to (contain) it. If all vertices have the same degree then it is the degree of the hypergraph. The hypergraph is c-regular if every vertex has the same degree c.

Let the set $V$ of vertices of an s-uniform hypergraph be partitioned into $t$ disjoint subsets $V_j$, $j = 1, 2, \ldots, t$. A hypergraph is said to be $t$-partite if no edge contains two vertices from the same set $V_j$, $j = 1, 2, \ldots, t$. 
3-partite, 3-uniform 4-regular hypergraph with 6 vertices and 8 hyperedges

Girth $g = 2$, girth of the Tanner graph $g_T = 4$. 
Incidence matrix of the hypergraph

\[ H = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & | & 0 & 0 & 0 & 0 \\
2 & 1 & 1 & 0 & 0 & | & 0 & 0 & 1 & 1 \\
3 & 1 & 0 & 1 & 0 & | & 0 & 1 & 0 & 1 \\
| & | & | & | & | & | & | & | & | \\
4 & 0 & 0 & 0 & 0 & | & 1 & 1 & 1 & 1 \\
5 & 0 & 0 & 1 & 1 & | & 1 & 1 & 0 & 0 \\
6 & 0 & 1 & 0 & 1 & | & 1 & 0 & 1 & 0 \\
\end{pmatrix} \]

This incidence matrix can be considered as a parity-check matrix of \((3, 4)\)-regular QC(tailbiting) LDPC block code with parent convolutional code determined by parity-check matrix

\[ H(D) = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & D & D \\
1 & D & 1 & D \\
\end{pmatrix} \]

of rate \( R = 1/4 \), tailbiting length \( M = 2 \), and minimum distance \( d_{\min} = 4 \).
Definition
Consider the additive group $(\Gamma, +)$, where $\Gamma = \{\gamma\}$. The corresponding voltage graph $G_V = \{E_B, V_B, \Gamma\}$ is obtained from the base graph $G_B = \{E_B, V_B\}$ by assigning a voltage value $\gamma(e, \xi, \xi')$ to every edge $e$ which connects the vertices $\xi$ and $\xi'$ and satisfies the property $\gamma(e, \xi, \xi') = -\gamma(e, \xi', \xi)$.

Definition
Let $G = \{E, V\}$ be a lifted graph obtained from a voltage graph $G_V$, where $E = E_B \times \Gamma$ and $V = V_B \times \Gamma$. Two vertices $(v, \gamma)$ and $(v', \gamma')$ are connected in the lifted graph by an edge if and only if $v$ and $v'$ are connected in the voltage graph $G_V$ with the voltage value of the corresponding edge given by $\gamma(e, v, v') = \gamma - \gamma'$. 
The definition of a cycle in a voltage graph is the same as for a regular graph except the additional restriction that neighboring edges may not connect the same nodes in reversed order.

**Voltage of a path** is the sum of all edge voltages involved. Note that even though the edges of a voltage graph are not directed, their edge voltage depends on the passing direction.

**Cycle in lifted graph** corresponds to a cycle in labeled (voltage) graph with zero sum of weights (voltages).
Large graphs from small graphs: lifting

$$H(D) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & D & 1 \\ 1 & D & D^3 \end{pmatrix} \quad W = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 3 \end{pmatrix}$$

Parent (voltage) graph and lifted graph

![Graphs](image)

- **a)** Base graph
- **b)** Lifted graph

$$g_B = 2, \quad g = 6.$$
Heawood graph as lifted graph

\[ g_B = 2, \ g = 6. \]
**Option 1:** Interpret $D$ as delay

\[ H(D) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & D & D^3 \end{pmatrix} = D^0 \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} + D^1 \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} + D^3 \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \]

\[ H(D) = D^0 H_0 + D^1 H_1 + D^2 H_2 + D^3 H_3 \]

In binary **TB** form

\[
H = \begin{pmatrix}
H_0 & H_1 & H_2 & H_3 \\
H_0 & H_1 & H_2 & H_3 \\
H_0 & H_1 & H_2 & H_3 \\
H_0 & H_1 & H_2 & H_3
\end{pmatrix}
\]
Option 2: Interpret $D$ as rotation

Binary incidence matrix of lifted graph $H = \begin{pmatrix} J_0 & J_0 & J_0 \\ J_0 & J_1 & J_3 \end{pmatrix};$

$$J = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$
Matrix representation of lifted graph

\[ H = \left( \begin{array}{ccc}
J^0 & J^0 & J^0 \\
J^0 & J^1 & J^3
\end{array} \right) \]

\[ = \left( \begin{array}{ccccccccccccccc}
1 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 1 \\
1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
| & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
| & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
| & | & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
| & | & | & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
| & | & | & | | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
| & | & | & | | | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
| & | & | & | | | | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & | & 1 & 1 & 1 \\
\end{array} \right) \]
The incidence matrix of the "utility" graph is

\[ H_g = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}. \]
By row permutations we can obtain a parity-check matrix

\[
H_g = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{pmatrix}.
\]

of a (9,3)-TB block code with parent convolutional determined by

\[
H_{\text{conv}}(D) = \begin{pmatrix}
1 & 1 & 1 \\
1 & D & D^2
\end{pmatrix}.
\]

or

\[
H = \begin{pmatrix}
1 & 1 & 1 \\
1 & \mathcal{J} & \mathcal{J}^2
\end{pmatrix} = \begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}.
\]
\[ H(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & D & D \\ 1 & D & 1 & D \end{pmatrix} \]

Tailbiting to length \( M = 2 \)

\[
H = \begin{pmatrix}
\begin{array}{cccc|cccc}
& v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\
\hline
c_0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
c_1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
c_2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
c_3 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
c_4 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
c_5 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
\end{array}
\end{pmatrix}
\]

Base matrix

\[ B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \]
Base Tanner graph
Polynomial matrix determining voltage graph

\[ H(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & D & D \\ 1 & D & 1 & D \\ 1 & D & 1 & D \end{pmatrix} \]
Polynomial matrix determining voltage graph

\[
H(D) = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & D & D \\
1 & D & 1 & D \\
\end{pmatrix}
\]
Polynomial matrix determining voltage graph

\[ H(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & D & D \\ 1 & D & 1 & D \end{pmatrix} \]
Polynomial matrix determining voltage graph

\[ H(D) = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & D & D \\
1 & D & 1 & D
\end{pmatrix} \]
Polynomial matrix determining voltage graph

\[ H(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & D & D \\ 1 & D & 1 & D \end{pmatrix} \]
Polynomial matrix determining voltage graph

\[ H(D) = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & D & D \\
1 & D & 1 & D
\end{pmatrix} \]
Polynomial matrix determining voltage graph

\[
H(D) = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & D & D \\
1 & D & 1 & D
\end{pmatrix}
\]
Polynomial matrix determining voltage graph

\[ H(D) = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & D & D \\
1 & D & 1 & D \\
\end{pmatrix} \]
QC-LDPC codes

\[ H(D) = \begin{pmatrix}
  1 & 1 & 1 & 1 \\
  1 & 1 & D & D \\
  1 & D & 1 & D \\
\end{pmatrix} \]

Tailbiting to length \( M = 2 \)

\[ H = \begin{pmatrix}
  v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\
  c_0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
  c_1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
  c_2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
  c_3 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
  c_4 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
  c_5 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
\end{pmatrix} \]
The $(c - b) \times c$ parity-check matrix of an $R = b/c$ QC LDPC LDPC code is

$$H(D) = \begin{pmatrix}
  h_{11}(D) & h_{12}(D) & \cdots & h_{1c}(D) \\
  h_{21}(D) & h_{22}(D) & \cdots & h_{2c}(D) \\
  \vdots & \vdots & \ddots & \vdots \\
  h_{(c-b)1}(D) & h_{(c-b)2}(D) & \cdots & h_{(c-b)c}(D)
\end{pmatrix}$$

where $h_{ij}(D)$ is either zero or monomial $D^{w_{ij}}$, where $w_{ij}$ is a nonnegative integer.

The degree matrix $W$ has $-1$s on zero positions of $H(D)$ and entries $w_{ij}$ on its nonnegative.

If every column and row of $H(D)$ contains $J$ and $K$ nonzero entries, respectively, the parent code is a $(J, K)$ regular LDPC code and irregular otherwise.
The integer matrix

\[ B = H(D) \big|_{D=1} \]

is called a **base** matrix. QC LDPC codes can be interpreted as labeling the base matrix by monomials.

\( B \) determines a **base** Tanner graph. The length of the shortest cycle in a Tanner graph of QC LDPC code is \( g \). The girth of a base Tanner graph determined by \( B \) is denoted \( g_B \).
Theorem
Cycle in lifted graph corresponds to a cycle in labeled (voltage) graph with zero sum.

Proof.
A path in the lifted graph passes vertices represented by pairs $(v, \gamma)$, $v \in V$, $\gamma \in \Gamma$. Consider path $v_1 \rightarrow v_2, \rightarrow \ldots \rightarrow v_n$. Its voltage is equal to sum of voltages $\gamma = (\gamma_1 - \gamma_2) + (\gamma_2 - \gamma_3) + \ldots + (\gamma_{n-1} - \gamma_n)$. If path is a cycle then $v_1 = v_n$, $\gamma_n = \gamma_1$, therefore the voltage of a cycle is $\gamma = 0$. □
Matrix representation of lifted graph

Binary incidence matrix of lifted graph

\[ H = \begin{pmatrix} \mathcal{I}^0 & \mathcal{I}^0 & \mathcal{I}^0 \\ \mathcal{I}^0 & \mathcal{I}^1 & \mathcal{I}^3 \end{pmatrix}; \]
Base Tanner graphs

Zero-sum cycle: \( a \xrightarrow{x} b \xrightarrow{-y} a \xrightarrow{z} b \xrightarrow{-x} a \xrightarrow{y} b \xrightarrow{-z} a \)

The voltage of the cycle is equal to zero for any \( x, y, \) and \( z \). Therefore, the maximum girth for voltage graph is equal to 6.

**Theorem**

*The girth of lifted graph cannot be larger than the length of the balanced cycle in the base graph.*
Balanced cycles

Balanced cycle lengths:

a) $g \leq \min\{\alpha + \beta, \alpha + \gamma, \beta + \gamma\}$

$$g_{bal} \geq 2\alpha + 2\beta + 2\gamma \geq 3g$$

b) $g \leq \min\{\alpha, \gamma\}$

$$g_{bal} \geq 2\alpha + 4\beta + 2\gamma \geq 4g$$
If shortest cycles overlap (theta-graph) then the equality is achieved if $\alpha = \beta = \gamma$ which is rather typical case because of symmetry of most good graphs. The special case of this scenario is Heawood graph where we have $\alpha = \beta = \gamma = 2$, and the length of balanced cycle is 6 which means that the maximum achievable girth of lifted graph is also equal to 6. Thus we conclude, that typically, lifting allows increasing the girth of the base graph approximately 3 times.
Theorem

If the parity-check matrix of base code contain submatrices of the shape

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\quad \text{or} \quad
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1
\end{bmatrix},
\]

then the lifted code cannot have its girth of Tanner graph larger than 12.

Proof.

The subgraphs corresponding to these fragments of base matrix have girth 2. For these fragments corresponding subgraphs of Tanner graph will have girth 4. Due to symmetry, in (??) we have equality, which means that the girth of balanced cycle is upperbounded by 12.
• Low density of parity checks facilitates low encoding and decoding complexity.

• LDPC codes can be described by (Tanner) graphs. Good codes correspond to graphs with large girth.

• Long graphs (codes) are obtained from short ones using lifting.

• Codes obtained by lifting from parent codes are called QC LDPC codes

• Girth cannot be large. Girth is proportional to log of number of vertices. Lifting allows increasing of girth at most 3 times.

• QC LDPC codes are used in many standards due to compact representation, and low encoding and decoding implementation complexity.
Encoding for LDPC code can be implemented by using matrix $G$. However, this method has quadratic complexity. The encoding complexity can be done with linear complexity in $H$ domain by imposing some restrictions on the shape of $H$.

$$H(D) = \{ h_{ij}(D) \} ,$$

$$h_{ij}(D) \in \{ 0, D^{w_{ij}} \} , i = 1, \ldots, c - b, j = 1, \ldots, c$$

where $w_{ij}$ are nonnegative integers.

Let

$$H(D) = (H_{bd}(D) \ h_0(D) \ H_{\text{inf}}(D))$$
Let

\[ H(D) = \begin{pmatrix} H_{bd}(D) & h_0(D) & H_{inf}(D) \end{pmatrix} \]

where \( H_{bd}(D) \) is a bidiagonal matrix of size \((c - b) \times (c - b - 1)\), \( h_0(D) \) is a column with at most 3 nonzero elements and \( H_{inf}(D) \) can be any monomial submatrix of the proper size. This submatrix corresponds to the information part of a codeword.

For example, if \( b = 4, c = 8 \),

\[
H(D) = \begin{pmatrix}
D^0 & 0 & 0 & D^\alpha & D^{w15} & D^{w16} & D^{w17} & D^{w18} \\
D^0 & D^0 & 0 & D^0 & D^{w25} & D^{w26} & D^{w27} & D^{w28} \\
0 & D^0 & D^0 & 0 & D^{w35} & D^{w36} & D^{w37} & D^{w38} \\
0 & 0 & D^0 & D^\alpha & D^{w45} & D^{w46} & D^{w47} & D^{w48}
\end{pmatrix}.
\]

where \( \alpha > 0 \).
Let codeword \( \mathbf{v} \) be

\[
\mathbf{v} = \begin{pmatrix} \mathbf{v}_{\text{check}} & \mathbf{v}_{\text{inf}} \end{pmatrix},
\]

where

\[
\mathbf{v}_{\text{inf}} = \begin{pmatrix} \mathbf{v}_{c-b+1} & \mathbf{v}_{c-b+2} & \cdots & \mathbf{v}_c \end{pmatrix}
\]

is the message sequence represented by \( b \) blocks of length \( M \) and

\[
\mathbf{v}_{\text{check}} = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_{c-b} \end{pmatrix}
\]

is the check part.

The two parts of the codeword \( \mathbf{v} \) have to satisfy

\[
\mathbf{v}_{\text{check}} \begin{pmatrix} H_{bd}(D) & h_0(D) \end{pmatrix}^T = \mathbf{v}_{\text{inf}} H_{\text{inf}}(D)^T
\]
Denote by

\[ s = (s_1 \ s_2 \ldots \ s_{c-b}) = v_{\text{inf}} H_{\text{inf}}(D)^T \]

a partial syndrome obtained by multiplying the information part of the codeword \( v \) by the parity-check matrix \( H(D) \).

Denote by

\[ e = (1 \ 1 \ldots \ 1) \]

the all-one vector of length \( M(c - b) \).
Multiplying both sides of (31) by $e^T$ and taking into account $H_{bd}(D)^T e^T = 0$ we obtain

$$v_{c-b} = \sum_{i=1}^{c-b} s_i$$ (1)

Once $v_{c-b}$ is known, first we modify the partial syndrome

$$\tilde{s} = s + v_{c-b} h_0(D)$$ (2)

and then all other parity-check bits can be found using the simple recursion

$$v_1 = \tilde{s}_1$$
$$v_i = \gamma_i^{-1}(\tilde{s}_i + \gamma_{i-1} v_{i-1}), \quad i = 2, \ldots, c - b$$ (4)

The encoding complexity grows linearly with the code length (more precisely, with the circulant size $M$).
The simplification of the encoding obtained due to specific shape of check part of \( H(D) \): all block-columns have even weight except the last block of parity part where weight-3 column \( h_0(D) \). By summing up all rows we can easily find the last block of check symbols. After that all other blocks can be easily computed recursively.
Notations: $\alpha$ and $\beta$ are hard decisions and reliabilities.

BP decoding:

$$
\alpha_i' \beta_i' = \alpha_i \beta_i + \sum_{i=1}^{J} \left( \prod_{h=1,h\neq i}^{K} \alpha_{jh} \right) f \left( \sum_{h=1,h\neq i}^{K} f(\beta_{jh}) \right), \quad (5)
$$

where

$$
f(\beta) = \ln \frac{e^\beta + 1}{e^\beta - 1} = -\ln \left( \tanh \left( \frac{\beta}{2} \right) \right), \quad \beta > 0
$$
Practical decoding algorithms

Sum-product or message passing algorithm:

• Horizontal step: For each parity check $c$, for all symbol nodes connected to $c$

$$L_{cv} = \left( \prod_{v'} \text{sign}(L_{cv'}) \right) f \left( \sum_{v' \neq v} f(L_{cv'}) \right), \quad (6)$$

• Vertical step For each symbol node $v$, for all check nodes connected to $v$

$$L_{cv} = y(v) + \sum_{c' \neq c} L_{c'v} \quad (7)$$

where $y(v)$ is input LLR for variable $v$, and sums and products are taken over nonzero elements of corresponding rows and columns.
function [hard, steps] = flooding(soft, maxsteps, V, C, dec_type)

% Inputs:
% soft is input soft-decision vector to be decoded
% maxsteps is maximum allowed number of iterations
% V is parity-check matrix in form of list of non-zero
%   positions
% in parity checks
% C is parity-check matrix in form of list of parity checks
% connected to each symbol position
% Outputs: hard decisions and true number of steps

n = length(soft);

r = size(V, 1);  % number of parity checks
rw = sum(V > 0, 2);  % row weights

r = size(V, 1);  % number of parity checks
rw = sum(V > 0, 2);  % row weights

r = size(C, 1);  % number of parity checks
rw = sum(C > 0, 2);  % column weights

Z = zeros(r, n);  % current LLRs

soft_out = soft;
Practical decoding algorithms

% Initialization
for i=1:r
  Z(i,V(i,1:rw(i))) = soft(V(i,1:rw(i)));
end

% Main loop
for steps=1:maxsteps
  for i=1:r % loop over checks
    p = V(i,1:rw(i)); % positions to process
    switch dec_type
      case 1, Z(i,p) = map_ms(Z(i,p));
      case 2, Z(i,p) = map_sp(Z(i,p));
    end
  end
  for i=1:n % symbol nodes
    p = C(i,1:cw(i));
    soft_out(i) = soft(i) + sum(Z(p,i));
    Z(p,i) = soft_out(i) - Z(p,i); % compute extrinsic
  end
  hard = soft_out < 0;
  if check_syndrome(hard,V) == 0, return; end
end
function soft_out=map_sp(y)
% LLR domain
hard=y<0;
synd=mod(sum(hard),2);
hard=mod(hard+synd,2);
alogpy=logexp(abs(y));
soft_out=(2*hard-1).*logexp(alogpy-sum(alogpy));

function y=logexp(x)
T=16; %19.07;
x(x>T)=T; x(x<-T)=-T;
y=log((exp(x)-1)./(exp(x)+1));

function [ws,syn]=check_syndrome(hard,V)
n=length(hard);
hard=[hard 0];
V(V==0)=n+1;
syn=mod(sum(hard(V),2),2);
ws=sum(syn);
Further simplifications are based on the following observation

- \( f(f(|x|)) = |x| \)
- \( f(|x|) \) is monotonically decreasing function of \( |x| \).

Using these arguments we obtain

\[
f \left( \sum_{v \not= v'} f(|L_{cv'}|) \right) \leq f \left( \max_{v \not= v'} f(|L_{cv'}|) \right) = \min_{v \not= v'} |L_{cv'}|.
\]

This approximation leads to min-sum (MS) version of the algorithm

\[
L_{cv} = \left( \prod_{v'} \text{sign}(L_{cv'}) \right) \min_{v' \not= v} |L_{cv'}|
\]
Furthermore, in order to make approximation tighter, additional parameters are introduced: $\alpha$ and $\beta$ are called normalization (scaling) factor and offset, respectively. Generalized formula for MS decoding is

$$L_{cv} = \left( \prod_{v'} \text{sign}(L_{cv'}) \right) \left( \alpha \min_{v' \neq v} |L_{cv'}| - \beta \right).$$

Important: With min-sum not only complex operations are excluded but also

- Memory saving: Not necessary to keep all LLRs. Keep only 2 minima
- The algorithm not critical to precision. It is enough to use 4-6 bits arithmetic precision.
function u = map_ms(y)
% offset = 0.4
hard = y < 0;
synd = mod(sum(hard), 2);
hard = (1 - 2 * mod(hard + synd, 2));
ay = abs(y);
[m1, i1] = min(ay); % first min
ay(i1) = 1000;
m2 = min(ay);
m1 = max(0, m1 - 0.4);
m2 = max(0, m2 - 0.4);
u = hard .* m1;
u(i1) = hard(i1) * m2;
function [hard, steps] = layered(soft, maxsteps, V, dec_type)

% V is code description in form of location
% nonzero positions in rows
n = length(soft); % code length
r = size(V, 1); % number of parity checks
rw = sum(V > 0, 2); % row weight
z = zeros(r, n); % tentative LLRs
hard = soft < 0; % hard decisions

for steps = 1:maxsteps
    for i = 1:r % loop over checks
        p = V(i, 1:rw(i)); % positions to process
        y = soft(p) - z(i, p); % deduct intrinsic
        switch dec_type
            case 1, u = map_ms(y); % min-sum version
            case 2, u = map_sp(y); % sum-prod version
        end
        soft(p) = u + y; % add extrinsic
        z(i, p) = u; % save tentative llr
    end
    hard(1:n) = soft < 0;
    if check Syndrome(hard, V) == 0, return; end
end