Coding Theory

(LDPC codes)

Lector: Nikolai L. Manev

Institute of Mathematics and Informatics, Sofia, Bulgaria

Historical remarks

- *Low-Density Parity-Check (LDPC)codes* were first proposed by Robert Gallager in 1962 (in “Low-Density Parity-Check Codes” IRE Trans. Inf. Theory, IT-8, Jan 1962, pp. 21-28). But, they were largely forgotten due to the lack of computational capability in those years and the rivalry of cyclic and convolutional codes intensively studied at that time.

Historical remarks

- *Low-Density Parity-Check (LDPC)codes* were first proposed by Robert Gallager in 1962 (in “Low-Density Parity-Check Codes” IRE Trans. Inf. Theory, IT-8, Jan 1962, pp. 21-28). But, they were largely forgotten due to the lack of computational capability in those years and the rivalry of cyclic and convolutional codes intensively studied at that time.
- LDPC codes were rediscovered by D.J.C. Mackay and R.M. Neal in 1996. (“Near Shannon Limit Performance of Low-Density Parity-Check Codes”, Electronics Letters, v. 32, Aug. 1996, pp. 1645-1646.) Using powerful modern computers David Mackay simulated their performance and thereby demonstrated their astonishing power.

Definition

Definition. An *LDPC code* is a linear $[n, n-m]$ code $\mathcal{C}$ whose $m \times n$ parity-check matrix $\mathbf{H}$ is sparse, that is, the density of nonzero entries in $\mathbf{H}$ is of the order of a small constant of n rather than nm.
Moreover, $\mathbf{H}$ should be pseudo-random, so that $\mathcal{C}$ will be a "random-like" code.

Definition. The code $\mathcal{C}$ is called *regular* (w_c, w_r) *LDPC code* if $\mathbf{H}$ contains exactly w_c nonzero entries in each column and w_r nonzero entries in each row, where $w_c < w_r$ and both are relatively small compared to n. Otherwise the code is called *irregular*.

The parameters n, w_r, w_c cannot be chosen independently, but must be related in $mw_r = nw_c$. Hence, the code rate of a regular LDPC code is $R = 1 - \frac{m}{n} = 1 - \frac{w_c}{w_r}$.

Example

In these lectures we will consider only binary LDPC codes.

Example. Consider a $(2, 4)$ regular LDPC code $\mathcal{C}$ of length 10, that is, with $w_c = 2$ ones in each column and $w_r = 4$ ones in each row of its parity-check matrix. The number of rows has to be $m = 20/4 = 5$. Here is such a matrix:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

This example is just for an illustration. $w_c \geq 3$ is necessary for good codes and usually their block length is very large, in the order of thousands of bits.

Graph representation

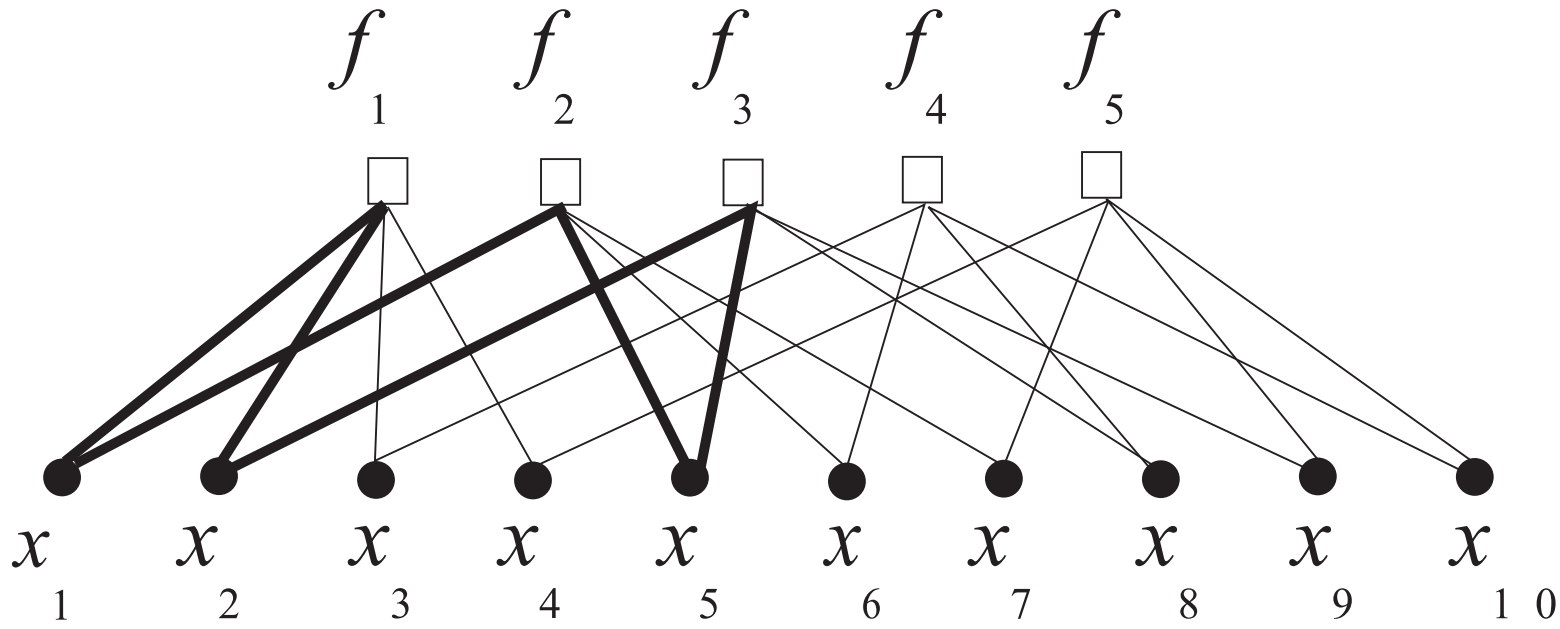
Definition. A *bipartite graph* is an undirected graph whose nodes (vertices) may be separated into two subsets, and where edges may only connect two nodes not residing in the same subset.

Any linear code defined by its parity-check matrix (in partial LDPC codes) can be specified by a bipartite graph. In coding theory it is referred to as *Tanner graph* after Michael Tanner who first proposed the bipartite graph as a tool of visualization of the parity-check equations.

The set of vertices of the Tanner graph corresponds to a $m \times n$ matrix $\mathbf{H}$ consists of m *check nodes*, one for each parity-check (i.e., row of $\mathbf{H}$) $f_1, f_2, \ldots, f_m$, and n *bit (or variable) nodes*, one for each codeword's bit $x_1, x_2, \ldots, x_n$. We designate check nodes with squares and bit nodes with circles.

Graph representation

An edge connects check node i to bit node j if and only if the i-th parity-check f_i involves the j-th bit x_j, i.e., $h_{ij} = 1$.



Tanner graph of a regular $(2, 4)$ LDPC code of length 10.

Any check node is connected to 4 (=row weight) edges and any variable node – to 2 (=column weight) edges. Total 20 edges.

How to construct LDPC codes

The first construction of LDPC codes is described by Gallager in his paper. Since 1996 many and quite various constructions have been proposed. We will not discuss these constructions and focuss only on algorithms for decoding LDPC codes. We will only note that Tanner graph is a tool for constructing such codes. Let the graph has M edges. Indexing their "check node" ends and "bit node" ends by the number $1, 2, \ldots, M$ the graph is uniquely defined by a permutation of the numbers $1, 2, \ldots, M$. For example, the permutation corresponding to the Tanner graph given on the previous slide transforms the bit node ends indexed by $1, 2, 3, \ldots, 20$ into the check node ends indexed respectively by

$$1, 5, 2, 9, 3, 13, 4, 17, 6, 10, 7, 14, 8, 18, 11, 15, 12, 19, 16, 20.$$

Encoding LDPC codes 1

The main drawback of the LDPC codes is their high encoding complexity. For encoding we need a parity-check matrix in a systematic form $\mathbf{H} = (\mathbf{P} \mid \mathbf{I}_m)$. Since the matrix $\mathbf{P}$ obtained after the transforming $\mathbf{H}$ will no longer be sparse, the expected number of XOR operations required by the encoding will be about $m(n-m)/2 = \frac{1}{2}R(1-R)n^2$, where R is the code rate. Hence, the complexity is quadric in the block length.

Richardson and Urbanke proposed a method that reduces the encoding complexity to $an^2 + O(n)$, where the constant factor a is very small and even large block length thus admit practically feasible encoders. ("Efficient encoding of Low-Density Parity-Check Codes", IEEE Transactions of Information Theory, vol. 47 (2001), no. 2, pp. 638-656.)

Encoding LDPC codes 2

The proposed method is based on the transforming $\mathbf{H}$ (the operation is called *preprocessing*) into the form

$$\mathbf{H} = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{pmatrix},$$

where $\mathbf{A}$ is $(m-g) \times (n-m)$, $\mathbf{B}$ is $(m-g) \times g$, $\mathbf{T}$ is $(m-g) \times (m-g)$, $\mathbf{C}$ is $g \times (n-m)$, $\mathbf{D}$ is $g \times g$, $\mathbf{E}$ is $g \times (m-g)$ and $\mathbf{T}$ is lower triangular with 1's along the diagonal. All these matrices are sparse, since the permutations preserve the property $\mathbf{H}$ to be sparse.

Decoding LDPC codes 1

Algorithms used to decode LDPC codes belong to a class of decoding algorithms which are collectively referred to as *message-passing* algorithms. Their name comes from the fact that these algorithms can be explained and visualized by the passing messages along the edges of a Tanner graph. The message-passing algorithms are *iterative* in sense that the messages pass forward and back between the bit and check nodes iteratively until a desired result is achieved or the process is stopped. Usually, the algorithms are named for the type of messages passed or the type of the operation performed at nodes. For example, in *bit-flipping* decoding, the messages are binary. In *belief propagation* decoding they are probabilities representing a level of belief about the values of the codeword bits.

The first decoding algorithm for LDPC code: Gallager.

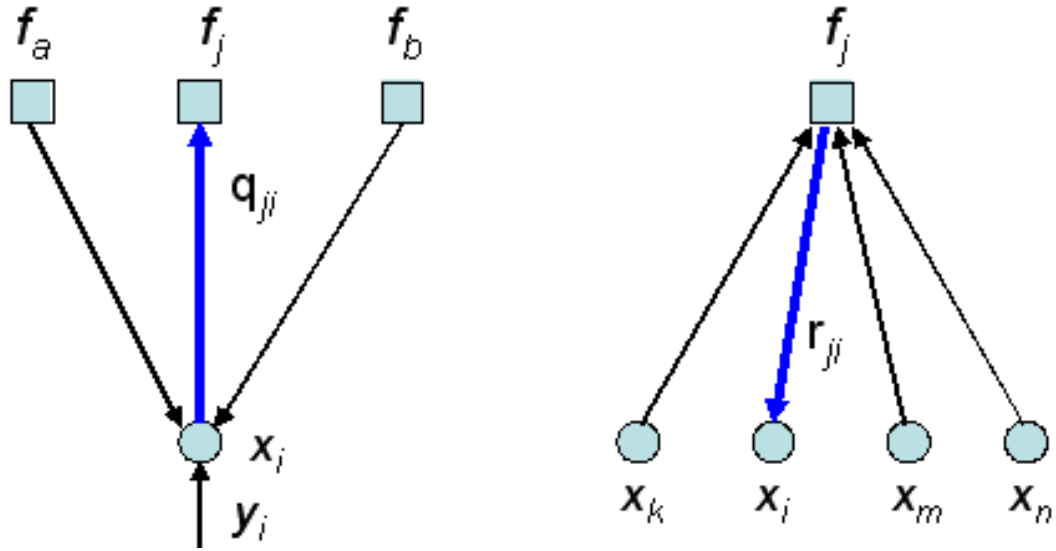
Decoding LDPC codes 2

Herein we will describe the *sum-product* (SP) algorithm. This is a soft decision message-passing algorithm. In contrast to the bit-flipping decoding that accepts an initial hard decision on the received bits as input, the sum-product algorithm accepts the probability of each received bit as input. The aim of SP decoding is to compute the *maximum a posteriori probability* (MAP) that a given bit, c_i, in the transmitted codewords, $\mathbf{c} = (c_1, c_2, \ldots, c_n)$, equals $b \in \{0, 1\}$, given the received vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. In each iteration the SP algorithm computes an approximation of the MAP value for each code bit c_i.

To illustrate the algorithm let us imagine that bit and check nodes represent two types of processors calculating some probabilities.

Decoding LDPC codes

In the first half iteration each bit node based on the information from input messages computes the probabilities of being equal to 0 and 1. In the next half iteration each check node processes its input messages and passes computed probabilities to its neighboring bit nodes (i.e., to nodes connected to it).



Note that each graph node works only having access to the information coming on the edges connected to it.

Decoding LDPC codes

Let

- $\mathbf{P}(j) = \{i \mid h_{ji} = 1\}$ be the set of indexes i such that the bits x_i participate in the j-th check f_j.
- $\mathbf{B}(i) = \{j \mid h_{ji} = 1\}$ be the set of indexes of those parity-checks that involve x_i.
- $\mathcal{P}_i$ be the event that the bits of $\mathbf{x}$ satisfy all the parity-checks involving x_i.
- $\mathcal{P}_i(j)$ be the event that the bits of $\mathbf{x}$ satisfy all the parity-checks involving x_i except the j-th check f_j (there is no information about the j-th check).

- $q_{ji}(b) = \Pr(x_i = b \mid \mathbf{y}, \boldsymbol{\mathcal{P}}_i(j)\}$ be the conditional probability that $x_i = b,\ b \in \{0, 1\}$, given the received vector $\mathbf{y}$ and information from all check nodes (i.e parity-check equations) except the j-th check node f_j.
- $r_{ji}(b) = \Pr(f_j = 0 \mid \mathbf{y}, x_i = b, \boldsymbol{\mathcal{B}}_j(i)\}$ be the conditional probability that the j-th parity-check equation is satisfied given that $x_i = b$, $\mathbf{y}$ is observed, and the probability mass functions $\{q_{j\nu}(0), q_{j\nu}(1)\},\ \nu \in \mathbf{P}(j) \setminus \{i\}$ of all bits involved in f_j other than x_i.

$$r_{ji}(0) = \frac{1}{2}\left(1 + \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} (1 - 2q_{j\nu}(1))\right)$$

$$r_{ji}(1) = \frac{1}{2}\left(1 - \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} (1 - 2q_{j\nu}(1))\right). \qquad (*)$$

Proposition (Gallager). Consider a sequence of m independent binary variables $\mathbf{a} = (a_1, a_2, ..., a_m)$ for which $\Pr(a_i = 1) = p_i$. Then the probabilities that $\mathbf{a}$ contains even number of 1's is

$$\frac{1}{2}[1 + \prod_{k=1}^{m}(1 - 2p_k)]$$

$$\begin{aligned} q_{ji}(b) &= \frac{\Pr(x_i = b, \mathbf{y}, \boldsymbol{\mathcal{P}}_i(j))}{\Pr(\mathbf{y}, \boldsymbol{\mathcal{P}}_i(j))} = \frac{\Pr(\boldsymbol{\mathcal{P}}_i(j) \mid x_i = b,\ \mathbf{y}) \Pr(x_i = b,\ \mathbf{y})}{\Pr(\mathbf{y}, \boldsymbol{\mathcal{P}}_i(j))} \\ &= \frac{\Pr(\boldsymbol{\mathcal{P}}_i(j) \mid x_i = b,\ \mathbf{y}) \Pr(x_i = b, \mid \mathbf{y}) \Pr(\mathbf{y})}{\Pr(\mathbf{y}) \Pr(\boldsymbol{\mathcal{P}}_i(j))} \\ &= K_{ji} \Pr(\boldsymbol{\mathcal{P}}_i(j) \mid x_i = b,\ \mathbf{y}) \Pr(x_i = b, \mid \mathbf{y}) \\ &= K_{ji} \Pr(x_i = b, \mid \mathbf{y}) \prod_{\nu \in \mathbf{B}(i) \setminus \{j\}} r_{ji}(b). \end{aligned}$$

The constants K_{ji} are determined from $q_{ji}(0) + q_{ji}(1) = 1$. Let us also set $P_i = \Pr(x_i = 1, | \, \mathbf{y})$. Then

$$q_{ji}(0) = K_{ji}(1 - P_i) \prod_{\nu \in \mathbf{B}(i) \setminus \{j\}} r_{\nu i}(0)$$

$$q_{ji}(1) = K_{ji} P_i \prod_{\nu \in \mathbf{B}(i) \setminus \{j\}} r_{\nu i}(1).$$

Let binary data are transmitted across AWGN channel using BPSK modulation, that is, the binary 0 is transmitted as $z = 1$ and the binary 1 as $z = -1$. Then $y_i = z_i + \eta_i$, $i = 1, 2, \ldots, n$, where the noise samples η_i are independent and normally distributed with zero mean and variance σ^2. Then

$$\Pr(y_i \mid z_i = \mp 1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_i \pm 1)^2}{2\sigma^2}}$$

Therefore,

$$P_i = \Pr(z_i = -1, \mid y_i) = \frac{\Pr(y_i \mid z_i = -1)\Pr(z_i = -1)}{\Pr(y_i)}$$

$$= \frac{\Pr(y_i \mid z_i = -1)/2}{\Pr(y_i \mid z_i = 1)/2 + \Pr(y_i \mid z_i = -1)/2} = \frac{1}{\left(1 + \frac{\Pr(y_i|z_i=1)}{\Pr(y_i|z_i=-1)}\right)}$$

(We assume as usually that $\Pr(z_i = 1) = \Pr(z_i = -1) = 1/2$.)

Hence, the initial (a priori) probabilities are

$$\Pr(x_i = 1, \mid \mathbf{y}) = P_i = \frac{1}{1 + e^{\frac{2y_i}{\sigma^2}}}, \qquad 1 - P_i = \frac{1}{1 + e^{-\frac{2y_i}{\sigma^2}}} .$$

Algorithm.

1. *Initialization*: For $i = 1, 2, \ldots, n$ set $q_{ji}(0) = 1 - P_i$ and $q_{ji}(1) = P_i$, for those j such that $h_{ji} = 1$.
2. Update $\{r_{ji}(b)\}$ using formulas $(*)$.
3. For $i = 1, 2, \ldots, n$ compute

$$Q_i(0) = K_i(1-P_i) \prod_{j \in \mathbf{B}(i)} r_{ji}(0), \quad Q_i(1) = K_i P_i \prod_{j \in \mathbf{B}(i)} r_{ji}(1),$$

   where K_i are such that $Q_i(0) + Q_i(1) = 1$.
4. Update $\{q_{ji}(b)\}$ by

$$q_{ji}(b) = K_{ji} Q_i(b) / r_{ji}(b), \ b = 0, 1,$$

   where K_{ji} are such that $q_{ji}(0) + q_{ji}(1) = 1$.

Algorithm (continuation).

5. For $i = 1, 2, \ldots, n$ set

$$x_i = \begin{cases} 0, & \text{if } Q_i(0) > Q_i(1) \\ 1, & \text{otherwise.} \end{cases}$$

6. If $\mathbf{x}\mathbf{H}^\tau = \mathbf{o}$ or some maximum number of iterations are performed, then stop, else go to step 2.

Decoding in Log-Domain

Recall that $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$. Hence, if $p_0 + p_1 = 1$, $p_i > 0$, then $\tanh(\frac{1}{2}\ln(p_0/p_1)) = p_0 - p_1 = 1 - 2p_1$. Therefore

$$\Delta q_{ji} = q_{ji}(0) - q_{ji}(1) = \tanh(L(q_{ji})/2)$$
$$\Delta r_{ji} = r_{ji}(0) - r_{ji}(1) = \tanh(L(r_{ji})/2).$$

where $L(q_{ji}) = \ln(q_{ji}(0)/q_{ji}(1))$ and $L(r_{ji}) = \ln(r_{ji}(0)/r_{ji}(1))$

Since $\Delta r_{ji} = \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \Delta q_{ji}$, then

$$\tanh(L(r_{ji})/2) = \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \tanh(|L(q_{j\nu})|/2) \times \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \epsilon_{j\nu},$$

where $\epsilon_{j\nu} = \mathrm{sign} L(q_{j\nu}) \in \{1, -1\}$.

Define the function $\Phi(x) \stackrel{\text{def}}{=} \ln \frac{e^x+1}{e^x-1} = -\ln\left(\tanh(\frac{x}{2})\right)$ for $x > 0$. $\Phi(x) > 0$ and $\Phi(x) = \Phi^{-1}(x) = 2\mathrm{arctanh}(e^{-x})$. Therefore

Decoding in Log-Domain

$$
\begin{aligned}
L(r_{ji}) &= \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \epsilon_{j\nu} \times 2\text{arctanh}\Big(\prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \tanh(|L(q_{j\nu})|/2)\Big) \\
&= \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \epsilon_{j\nu} \times 2\text{arctanh}\Big(\exp\Big(\ln\Big(\prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \tanh(|L(q_{j\nu})|/2)\Big)\Big)\Big) \\
&= \prod_{\nu \in \mathbf{P}(j)\setminus\{i\}} \epsilon_{j\nu} \times \Phi\left(\sum_{\nu \in \mathbf{P}(j)\setminus\{i\}} \Phi(|L(q_{j\nu})|)\right). \qquad (**)
\end{aligned}
$$

Algorithm.

1. *Initialization*: For $i = 1, 2, \ldots, n$ set $L(q_{ji}) = \ln \frac{1-P_i}{P_i} = \frac{2y_i}{\sigma^2}$ for those j such that $h_{ji} = 1$.
2. Update $\{L(r_{ji})\}$ using formulas $(**)$.
3. For $i = 1, 2, \ldots, n$ compute
$$L(Q_i) = \frac{2y_i}{\sigma^2} + \sum_{j \in \mathbf{B}(i)} L(r_{ji}).$$
4. Update $\{L(q_{ji})\}$ by $L(q_{ji}) = L(Q_i) - L(r_{ji})$.
5. For $i = 1, 2, \ldots, n$ set $x_i = \begin{cases} 0, & \text{if } L(Q_i) > 0 \\ 1, & \text{otherwise.} \end{cases}$
6. If $\mathbf{x}\mathbf{H}^T = \mathbf{o}$ or some maximum number of iterations are performed, then stop, else go to step 2.

Examples 1

Consider the $(2, 4)$ regular LDPC code $\mathcal{C}$ of length 10 defined on page 4 (slide 5). Assume that BPSK modulation is used and the communication channel is an AWGN channel with SNR=3dB. Let the codeword $\mathbf{c} = (1010010000)$, that is the set of real values $[-1, 1, -1, 1, 1, -1, 1, 1, 1, 1]$, is sent across the channel. After a simulation of the AWGN channel by Matlab the vector $\mathbf{y}$ with real entries is received:

$\mathbf{y}$ =(-2.1356, 1.1822, -1.7479, 2.0018, 0.4300, -0.6257, 1.1553, 0.3473, -0.5367, 0.9581).

Hence in the case of hard decision detection the output vector will have an error in 9th position.

The added Gaussian noise corresponding to the required SNR=3dB has variance $\sigma^2 = 10^{-0.3}$ Therefore, the vector of initial values $\ln \frac{1-P_i}{P_i}, i = 1, \ldots, 10,$ is $\mathbf{LPI} = 2\mathbf{y}/10^{-0.3} =$

(-8.5222, 4.7176, -6.9750, 7.9882, 1.7159, -2.4969, 4.6103, 1.3859, -2.1417, 3.8233).

Examples

For the sake of convenience let us write the sets $B(i),\ i = 1, 2, \ldots, 10,$ as columns of the matrix

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 4 \\ 2 & 3 & 4 & 5 & 3 & 4 & 5 & 4 & 5 & 5 \end{pmatrix}$$

Similarly, let the rows of the matrix

$$\mathbf{P} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 5 & 6 & 7 \\ 2 & 5 & 8 & 9 \\ 3 & 6 & 8 & 10 \\ 4 & 7 & 9 & 10 \end{pmatrix}$$

be the sets $P(j),\ j = 1, 2, 3, 4, 5.$

Let

$$\mathbf{Lq} = \begin{pmatrix} lq_{11} & lq_{12} & lq_{13} & lq_{14} & lq_{25} & lq_{26} & lq_{27} & lq_{38} & lq_{39} & lq_{4,10} \\ lq_{21} & lq_{32} & lq_{43} & lq_{54} & lq_{35} & lq_{46} & lq_{57} & lq_{48} & lq_{59} & lq_{5,10} \end{pmatrix}$$

be the matrix whose entries are the values $lq_{ji} = L(q_{ji})$ arranged such that $j \in \mathbf{B}(i)$.

Similarly, let the values $L(r_{ji})$ be written as entries of the matrix

$$\mathbf{Lr} = \begin{pmatrix} lr_{11} & lr_{12} & lr_{13} & lr_{14} & lr_{25} & lr_{26} & lr_{27} & lr_{38} & lr_{39} & lr_{4,10} \\ lr_{21} & lr_{32} & lr_{43} & lr_{54} & lr_{35} & lr_{46} & lr_{57} & lr_{48} & lr_{59} & lr_{5,10} \end{pmatrix}$$

Initialization. The initial value of $\mathbf{Lq}$ is

$$\begin{matrix} -8.5222 & 4.7176 & -6.9750 & 7.9882 & 1.7159 & -2.4969 & 4.6103 & 1.3859 & -2.1417 & 3.8233 \\ -8.5222 & 4.7176 & -6.9750 & 7.9882 & 1.7159 & -2.4969 & 4.6103 & 1.3859 & -2.1417 & 3.8233 \end{matrix}$$

Round 1.

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 \end{pmatrix}$$

$\mathbf{Lr} =$

$$\begin{matrix} -4.5843 & 6.5202 & -4.6591 & 4.5981 & 2.3815 & -1.6628 & 1.3529 & -1.2064 & 0.8704 & 1.1191 \\ -1.3183 & -0.6842 & -1.0638 & -1.9059 & -1.0083 & -1.3044 & -1.9712 & 2.2544 & 3.4378 & -2.0590 \end{matrix}$$

$\mathbf{LQ} =$

$$-14.4248 \quad 10.5536 \quad -12.6980 \quad 10.6805 \quad 3.0891 \quad -5.4641 \quad 3.9919 \quad 2.4339 \quad 2.1664 \quad 2.8835$$

These entries of $\mathbf{LQ}$ give $\mathbf{c} = (1010010000)$, which is a codeword. We thus stop.

Examples

Let now the received vector $\mathbf{y}$ be equal to:

$\mathbf{y}$ =(-0.7916, 0.0541, -0.4943, 2.1494, 0.5103, -0.3926, 1.8878, -0.1283, -0.0201, 1.4043).

Hence, in the case of hard decision detection, the output vector will have errors in 8th and 9th positions.

The added Gaussian noise corresponding to the required SNR=3dB has variance $\sigma^2 = 10^{-0.3}$ Therefore, the vector of initial values $\ln \frac{1-P_i}{P_i}, i = 1, \ldots, 10,$ is $\mathbf{LPI} = 2\mathbf{y}/10^{-0.3} =$

(-3.1589, 0.2159, -1.9725, 8.5772, 2.0364, -1.5667, 7.5333, -0.5120, -0.0802, 5.6039).

The initial value of $\mathbf{Lq}$ is

$$\begin{array}{cccccccccc} -3.1589 & 0.2159 & -1.9725 & 8.5772 & 2.0364 & -1.5667 & 7.5333 & -0.5120 & -0.0802 & 5.6039 \\ -3.1589 & 0.2159 & -1.9725 & 8.5772 & 2.0364 & -1.5667 & 7.5333 & -0.5120 & -0.0802 & 5.6039 \end{array}$$

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \end{pmatrix}$$

Round 1.

$$\mathbf{Lr} =$$

$$\begin{matrix} -0.1628 & 1.7109 & -0.1981 & 0.1496 & 1.3883 & -1.7571 & 1.0009 & -0.0066 & -0.0414 & -0.2492 \\ -1.1065 & 0.0154 & 0.3285 & -0.0795 & 0.0022 & 0.3804 & -0.0796 & 1.0750 & 5.4246 & -0.0801 \end{matrix}$$

$$\mathbf{LQ} =$$

$$\begin{matrix} -4.4282 & 1.9423 & -1.8421 & 8.6473 & 3.4268 & -2.9434 & 8.4546 & 0.5564 & 5.3030 & 5.2746 \end{matrix}$$

This entries of $\mathbf{LQ}$ give $\mathbf{x} = [-1, 1, -1, 1, 1, -1, 1, 1, 1, 1]$, i.e., $\mathbf{c} = (1010010000)$, which is a codeword and we stop.

Note that we decoded 2 errors!!

Indeed, our code is an 1-error correcting $[10, 6, 3]$ shorten Hamming code.

Examples

Let now the received vector $\mathbf{y}$ be equal to:

$\mathbf{y}$ =(0.0996, 1.5012, 0.3857, 1.3572, 2.3200, -1.2406, 0.1931, 0.8505, 1.8426, 0.2098).

Hence in the case of hard decision detection, the output vector will have errors in first and third positions.

The added Gaussian noise corresponding to the required SNR=3dB has variance $\sigma^2 = 10^{-0.3}$ Therefore, the vector of initial values $\ln \frac{1-P_i}{P_i}, i = 1, \ldots, 10,$ is $\mathbf{LPI} = 2\mathbf{y}/10^{-0.3} =$

(0.3975, 5.9906, 1.5391, 5.4159, 9.2580, -4.9506, 0.7706, 3.3939, 7.3529, 0.8372).

The initial value of $\mathbf{Lq}$ is

$$\begin{array}{cccccccccc} 0.3975 & 5.9906 & 1.5391 & 5.4159 & 9.2580 & -4.9506 & 0.7706 & 3.3939 & 7.3529 & 0.8372 \\ 0.3975 & 5.9906 & 1.5391 & 5.4159 & 9.2580 & -4.9506 & 0.7706 & 3.3939 & 7.3529 & 0.8372 \end{array}$$

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Round 1.

$$\mathbf{Lr} =$$

$$\begin{matrix} 1.5088 & 0.2528 & 0.3918 & 0.2538 & -0.1423 & 0.1443 & -0.3916 & 5.7327 & 3.3195 & -1.3744 \\ -0.7585 & 3.3723 & -0.7649 & 0.2924 & 3.3046 & 0.4881 & 0.8277 & -0.5158 & 0.2902 & 0.7620 \end{matrix}$$

$$\mathbf{LQ} =$$

$$\begin{matrix} 1.1477 & 9.6156 & 1.1661 & 5.9621 & 12.4203 & -4.3183 & 1.2067 & 8.6109 & 10.9626 & 0.2248 \end{matrix}$$

This entries of $\mathbf{LQ}$ give $\mathbf{x} = [1, 1, 1, 1, 1, -1, 1, 1, 1, 1]$, i.e., $\mathbf{c} = (0000010000)$, which is not a codeword and we update $\mathbf{Lq}$ and return to step 2.

Round 2.

$$\mathbf{Lq} =$$

$$\begin{matrix} -0.3610 & 9.3629 & 0.7742 & 5.7083 & 12.5626 & -4.4626 & 1.5983 & 2.8781 & 7.6431 & 1.5992 \\ 1.9062 & 6.2434 & 1.9310 & 5.6697 & 9.1157 & -4.8063 & 0.3789 & 9.1267 & 10.6725 & -0.5371 \end{matrix}$$

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \end{pmatrix}$$

$$\mathbf{Lr} =$$

$$\begin{matrix} 0.7684 & -0.1311 & -0.3585 & -0.1319 & -1.0476 & 1.0769 & -1.8332 & 5.9785 & 2.8424 & -1.8766 \\ -1.5452 & 2.8678 & -1.5607 & -0.0983 & 2.8361 & 1.0868 & -0.5332 & -1.0659 & -0.0976 & 0.3762 \end{matrix}$$

$$\mathbf{LQ} =$$

$$\begin{matrix} -0.3793 & 8.7273 & -0.3801 & 5.1857 & 11.0466 & -2.7869 & -1.5959 & 8.3065 & 10.0977 & -0.6632 \end{matrix}$$

This entries of $\mathbf{LQ}$ give $\mathbf{x} = [-1, 1, -1, 1, 1, -1, -1, 1, 1, -1]$, i.e., $\mathbf{c} = (1010011001)$, which is not a codeword. Hence, we update $\mathbf{Lq}$ and return to step 2.

Round 3.

$$\mathbf{Lq} =$$

$$\begin{matrix} -1.1477 & 8.8583 & -0.0215 & 5.3176 & 12.0941 & -3.8638 & 0.2373 & 2.3280 & 7.2553 & 1.2135 \\ 1.1659 & 5.8595 & 1.1806 & 5.2840 & 8.2105 & -3.8737 & -1.0626 & 9.3725 & 10.1953 & -1.0394 \end{matrix}$$

Examples

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

$$\mathbf{Lr} =$$

$$\begin{matrix} -0.0213 & 0.0111 & -1.1335 & 0.0112 & -0.1190 & 0.1241 & -1.1072 & 5.5647 & 2.2968 & -1.1213 \\ -0.2275 & 2.3181 & -1.1518 & 0.4731 & 2.2925 & 0.5910 & -1.0269 & -0.5657 & 0.4681 & -1.0497 \end{matrix}$$

$$\mathbf{LQ} =$$

$$0.1487 \quad 8.3198 \quad -0.7462 \quad 5.9002 \quad 11.4315 \quad -4.2355 \quad -1.3635 \quad 8.3929 \quad 10.1179 \quad -1.3338$$

This entries of $\mathbf{LQ}$ give $\mathbf{x} = [1, 1, -1, 1, 1, -1, -1, 1, 1, -1]$, i.e., $\mathbf{c} = (0010011001)$, which is not a codeword. Hence, we update $\mathbf{Lq}$ and return to step 2.

Round 4.

$$\mathbf{Lq} =$$

$$\begin{matrix} 0.1700 & 8.3087 & 0.3874 & 5.8890 & 11.5505 & -4.3597 & -0.2563 & 2.8283 & 7.8211 & -0.2125 \\ 0.3761 & 6.0016 & 0.4056 & 5.4271 & 9.1390 & -4.8265 & -0.3367 & 8.9586 & 9.6497 & -0.2840 \end{matrix}$$

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

$\mathbf{Lr} =$

$0.3850 \;\; 0.0323 \;\; 0.1689 \;\; 0.0324 \;\; 0.0462 \;\; -0.0474 \;\; -0.3664 \;\; 5.8147 \;\; 2.7857 \;\; -0.3989$

$0.2497 \;\; 2.8197 \;\; 0.2090 \;\; 0.0471 \;\; 2.7809 \;\; -0.0424 \;\; -0.2815 \;\; 0.0417 \;\; 0.0466 \;\; -0.3336$

$\mathbf{LQ} =$

$1.0322 \;\; 8.8426 \;\; 1.9171 \;\; 5.4954 \;\; 12.0851 \;\; -5.0404 \;\; 0.1227 \;\; 9.2504 \;\; 10.1853 \;\; 0.1047$

This entries of $\mathbf{LQ}$ give $\mathbf{x} = [1, 1, 1, 1, 1, -1, 1, 1, 1, 1]$, i.e., $\mathbf{c} = (0000010000)$, which is not a codeword. Hence, we update $\mathbf{Lq}$ and return to step 2.

Round 5.

$\mathbf{Lq} =$

$0.6472 \;\; 8.8103 \;\; 1.7482 \;\; 5.4630 \;\; 12.0389 \;\; -4.9930 \;\; 0.4891 \;\; 3.4356 \;\; 7.3996 \;\; 0.5036$

$0.7824 \;\; 6.0228 \;\; 1.7081 \;\; 5.4484 \;\; 9.3042 \;\; -4.9980 \;\; 0.4042 \;\; 9.2087 \;\; 10.1386 \;\; 0.4383$

Examples

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

$$\mathbf{Lr} =$$

$$\begin{matrix} 1.7240 & 0.4434 & 0.6411 & 0.4472 & -0.1766 & 0.1791 & -0.7708 & 5.7682 & 3.3606 & -1.6722 \\ -0.4822 & 3.4141 & -0.4965 & 0.0861 & 3.3457 & 0.3452 & 0.4344 & -0.3406 & 0.0853 & 0.4006 \end{matrix}$$

$$\mathbf{LQ} =$$

$$\begin{matrix} 1.6393 & 9.8481 & 1.6838 & 5.9492 & 12.4271 & -4.4264 & 0.4342 & 8.8216 & 10.7988 & -0.4344 \end{matrix}$$

This entries of $\mathbf{LQ}$ give $\mathbf{x} = [1, 1, 1, 1, 1, -1, 1, 1, 1, -1]$, i.e., $\mathbf{c} = (0000010001)$, which is not a codeword. Hence, we update $\mathbf{Lq}$ and return to step 2.

Round 6.

$$\mathbf{Lq} =$$

$$\begin{matrix} -0.0848 & 9.4046 & 1.0427 & 5.5020 & 12.6037 & -4.6054 & 1.2050 & 3.0534 & 7.4383 & 1.2378 \\ 2.1215 & 6.4340 & 2.1803 & 5.8631 & 9.0814 & -4.7716 & -0.0002 & 9.1622 & 10.7135 & -0.8350 \end{matrix}$$

Examples

$$\mathbf{sq} = sign(\mathbf{Lq}) = \begin{pmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

$$\mathbf{Lr} =$$

$$\begin{matrix} 1.0324 & -0.0402 & -0.0841 & -0.0406 & -0.8835 & 0.9039 & -2.0426 & 6.0713 & 3.0177 & -2.1081 \\ -1.1751 & 3.0387 & -1.2111 & 0.0001 & 3.0080 & 0.9407 & -0.8297 & -0.9228 & 0.0001 & -0.0002 \end{matrix}$$

$$\mathbf{LQ} =$$

$$\begin{matrix} 0.2547 & 8.9890 & 0.2439 & 5.3755 & 11.3825 & -3.1060 & -2.1017 & 8.5425 & 10.3707 & -1.2712 \end{matrix}$$

The entries of $\mathbf{LQ}$ give $\mathbf{x} = [1, 1, 1, 1, 1, -1, -1, 1, 1, -1]$, i.e., $\mathbf{c} = (0000011001)$, which is a codeword and we stop.

Note that the decoding is wrong!! The obtained codeword $\mathbf{c} = (0000011001)$ differs from the sent codeword $\mathbf{c} = (1010010000)$!!

The end of the part

Thank You for Attention!