4F5: Advanced Communications and Coding

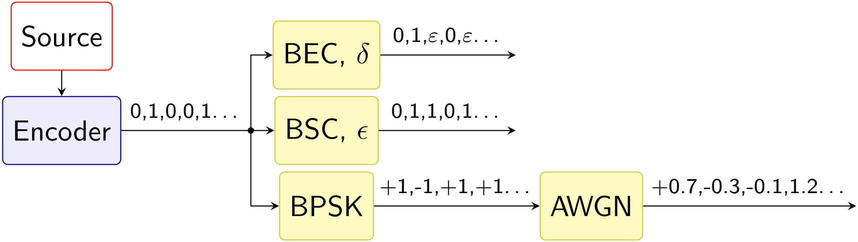
Coding Handout 3: Low Density Parity Check (LDPC) Codes

Jossy Sayir

Signal Processing and Communications Lab
Dept. of Engineering
jossy.sayir@eng.cam.ac.uk

Michaelmas Term 2014

Probabilistic Decoding



- all channels are memoryless and can be described by two (marginal) conditional distributions $P_{Y|X}(.|0)$ and $P_{Y|X}(.|1)$
- the output alphabet is different for every channel
- the "matrix inversion" decoder for linear codes will only work for the BEC

Can all these channels be treated in a common framework?

A-Posteriori Probabilities (APPs)

For each received symbol y_k, compute the probability distribution

$$\begin{cases} P_{X|Y}(0|y_k) \\ P_{X|Y}(1|y_k) \end{cases}$$

BEC

$$\begin{cases} y_k = 0 \rightarrow P_{X|Y}(0|y_k) = 1 \\ y_k = 1 \rightarrow P_{X|Y}(1|y_k) = 1 \\ y_k = \varepsilon \rightarrow P_{X|Y}(0|y_k) = P_{X|Y}(1|y_k) = 1/2 \end{cases}$$

BSC (supposing $P_X(0) = P_X(1) = 1/2$)

$$\begin{cases} y_k = 0 \rightarrow P_{X|Y}(0|y_k) = \frac{P_{Y|X}(y_k|0)P_X(0)}{P_Y(y_k)} = \frac{(1-\epsilon)1/2}{1/2} = 1-\epsilon \\ y_k = 1 \rightarrow P_{X|Y}(0|y_k) = \frac{P_{Y|X}(y_k|0)P_X(0)}{P_Y(y_k)} = \frac{\epsilon \cdot 1/2}{1/2} = \epsilon \end{cases}$$

A-Posteriori Probabilities (APPs)

AWGN with BPSK modulation

$$
\begin{aligned}
P_{X|Y}(+1|y_k) &= \frac{p_{Y|X}(y_k|+1)P_X(+1)}{p_{Y|X}(y_k|+1)P_X(+1) + p_{Y|X}(y_k|-1)P_X(-1)} \\
&= \frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-1)^2}{2\sigma^2}} \cdot 1/2}{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-1)^2}{2\sigma^2}} \cdot 1/2 + \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y+1)^2}{2\sigma^2}} \cdot 1/2} \\
&= \frac{1}{1 + e^{\frac{(y-1)^2-(y+1)^2}{2\sigma^2}}} = \frac{1}{1 + e^{\frac{-2y}{\sigma^2}}} \\
P_{X|Y}(-1|y_k) &= \frac{1}{1 + e^{\frac{2y}{\sigma^2}}}
\end{aligned}
$$

$\rightarrow$ all channels look the same in the language of a-posteriori probabilities (a "message" consisting of two probabilities for 0 and 1 for each received symbol)

Log Likelihood Ratios

Probabilities can be unpleasant to work with:

- two numbers that sum to one,
- good error performance means that we'll be handling numbers very close to zero or one.

Log likelihood ratios (LLRs)

Likelihood is defined as $\lambda(y_k|x) = P_{Y|X}(y_k|x)$. Likelihood ratio is $\Lambda(y_k) = \lambda(y_k|0)/\lambda(y_k|1)$ and the log likelihood ratio is the logarithm of the likelihood, i.e.

$$L(y_k) = \log \frac{P_{Y|X}(y_k|0)}{P_{Y|X}(y_k|1)}$$

Log Likelihood Ratios for Binary Input Channels with Equiprobable Inputs

For binary input channels with equiprobable inputs $P_X(0) = P_X(1) = 1/2$, it is easy to show that the log-likelihood ratio is also the log ratio of a-posteriori probabilities.

LLRs are also log(APPs)

$$L(y_k) = \log \frac{P_{Y|X}(y_k|0)}{P_{Y|X}(y_k|1)} = \log \frac{P_{X|Y}(0|y_k)}{P_{X|Y}(1|y_k)}$$

Conversions from LLRs to APPs

$$e^{L(y_k)} = \frac{P_{X|Y}(0|y_k)}{P_{X|Y}(1|y_k)} = \frac{P_{X|Y}(0|y_k)}{(1 - P_{X|Y}(0|y_k))}$$

hence

$$P_{X|Y}(0|y_k) = \frac{1}{1 + e^{-L(y_k)}} \text{ and } P_{X|Y}(1|y_k) = \frac{1}{1 + e^{L(y_k)}}$$

Log Likelihood Ratios for a few channels of interest

BEC

$$\begin{cases} y_k = 0 \rightarrow L(y_k) = \log \frac{1}{0} = +\infty \\ y_k = \varepsilon \rightarrow L(y_k) = \log \frac{1/2}{1/2} = 0 \\ y_k = 1 \rightarrow L(y_k) = \log \frac{0}{1} = -\infty \end{cases}$$

BSC

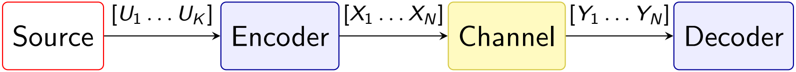
$$\begin{cases} y_k = 0 \rightarrow L(y_k) = \log \frac{1-\epsilon}{\epsilon} = L_\epsilon \\ y_k = 1 \rightarrow L(y_k) = \log \frac{\epsilon}{1-\epsilon} = -L_\epsilon \end{cases}$$

Log Likelihood Ratios for a few channels of interest

AWGN with BPSK modulation

$$
\begin{aligned}
L(y) &= \log \frac{p_{Y|X}(y|+1)}{p_{Y|X}(y|-1)} \\
&= \log \frac{\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(y-1)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(y+1)^2}{2\sigma^2}}} \\
&= \log e^{\frac{(y-1)^2-(y+1)^2}{2\sigma^2}} = \frac{2}{\sigma^2} y
\end{aligned}
$$

Maximum Likelihood Decoding



Maximum A-posteriori (MAP) decoder: optimal decoder picks the most probable source sequence $U_1 \dots U_K$, or, equivalently, the most probable codeword $X_1 \dots X_N$ given the observed sequence $X_1 \dots X_N$

Maximum Likelihood (ML) decoder: when all codewords are equally likely, as is the case for a linear code, we can equivalently pick the codeword whose likelihood is maximised, i.e., the ML decoder picks a codeword in a codebook $\mathcal{C}$ such that

$$\hat{x}_1 \dots \hat{x}_N = \mathrm{argmax}_{x_1 \dots x_N \in \mathcal{C}} P(y_1 \dots y_N | x_1 \dots x_N)$$

Symbol-wise (soft) Decoding

- ML and MAP sequence decoding is optimal in the “block error rate” regime
- What is the optimal decoding rule for “bit error rate” (or “symbol error rate” for non-binary codes)?

Bitwise MAP Rule

For a given received sequence $y_1, \ldots, y_N$,

$$\hat{u}_i = \text{argmax}_{u \in \{0,1\}} P_{U_i|Y_1 \ldots Y_N}(u|y_1 \ldots y_N) \text{ for } i = 1, \ldots, K$$

Symbol-wise MAP Decoding for Systematic Codes

For systematic codes, $X_1, \ldots, X_K = U_1, \ldots, U_K$ and hence the pevious expression can be replaced by

$$\hat{x}_i = \text{argmax}_{x \in \{0,1\}} P_{X_i | Y_1, \ldots, Y_N}(x | y_1, \ldots, y_n)$$

where, for convenience, we will use this expression for $i = 1, \ldots, N$ even though we are only truly interested in $i = 1, \ldots, K$.

Optimal Systematic Decoder

The optimal "bit error rate" decoder boils down to estimating each individual code symbol given the complete vector of observations.

Note that the resulting estimated sequence $\hat{x_1}, \ldots, \hat{x_N}$ is not necessarily a codeword.

Symbol-wise MAP Decoding for Systematic Codes

Without loss of generality, let us consider decoding of the first code symbol X_1,

$$P_{X_1|Y_1\ldots Y_N}(0|y_1\ldots y_N) = \frac{1}{P_{Y_1\ldots Y_N}(y_1\ldots y_N)} P_{X_1,Y_1\ldots Y_N}(0,y_1\ldots y_N)$$
$$= \alpha \sum_{\substack{x_1\ldots x_N\in\mathcal{C}\\ x_1=0}} P_{X_1\ldots X_N,Y_1\ldots Y_N}(x_1\ldots x_N, y_1\ldots y_N)$$
$$= \beta \sum_{\substack{x_1\ldots x_N\in\mathcal{C}\\ x_1=0}} \prod_{i=1}^{N} P_{X|Y}(x_i|y_i)$$

For each code symbol to be estimated, we need to take a sum over all codewords that have a zero in the corresponding position.

$\longrightarrow$ Not a practical decoder!

(For linear codes, supposing the first column of G contains d ones, we still need to sum over $2^{d-1}\cdot 2^{K-d} = 2^{K-1}$ possibilities, which is prohibitively large.)

Two curious codes

Repetition (or repeat) Code

A repetition code is an $(N, 1)$ linear systematic code where the data symbol is repeated N times. Its generator and parity-check matrices are

$$G = [1, 1, \ldots, 1] \text{ and } H = \begin{bmatrix} 1 & 1 & 0 & \ldots & 0 \\ 1 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \ldots & 1 \end{bmatrix}$$

Single Parity-Check (SPC) Code

A single parity-check code is a linear systematic code where a single parity-check is added at the end of the data word. Its generator and parity=check matrices are

$$G = \begin{bmatrix} 1 & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 1 \end{bmatrix} \text{ and } H = [1, 1, \ldots, 1]$$

Symbol-wise decoding for repetition codes

The sum over all codewords with a zero at position 1 becomes just one term for the all-zero codeword, i.e.,

$$
\begin{aligned}
P_{X_1|Y_1\ldots Y_N}(0|y_1\ldots y_N) &= \beta \sum_{\substack{x_1\ldots x_N\in\mathcal{C}\\ x_1=0}} \prod_{i=1}^{N} P_{X|Y}(x_i|y_i) \\
&= \beta \prod_{i=1}^{N} P_{X|Y}(0|y_i)
\end{aligned}
$$

and similarly $P_{X_1|Y_1\ldots Y_N}(1|y_1\ldots y_N) = \beta \prod_{i=1}^{N} P_{X|Y}(1|y_i)$ for the all-one codeword.

Summary: decoding for the repetition code

Take the product of $P_{X|Y}(0|y_i)$ for all received symbols, then the product of $P_{X|Y}(1|y_i)$, then divide the two products obtained by their sum to normalise and get rid of the β in the expression.

Symbol-wise decoding for repetition codes in the log-likelihood domain

$$
\begin{aligned}
\log \frac{P_{X_1|Y_1\ldots Y_N}(0|y_1\ldots y_N)}{P_{X_1|Y_1\ldots Y_N}(1|y_1\ldots y_N)} &= \log \frac{\beta\prod_{i=1}^{N} P_{X|Y}(0|y_i)}{\beta\prod_{i=1}^{N} P_{X|Y}(1|y_i)} \\
&= \sum_{i=1}^{N} \log \frac{P_{X|Y}(0|y_i)}{P_{X|Y}(1|y_i)} \\
&= \sum_{i=1}^{N} L(y_i)
\end{aligned}
$$

Decoding in the log domain

The resulting LLR is the sum of the LLRs from the channel.

Special Case BEC

The product $\beta \prod_{i=1}^{N} P_{X|Y}(0|y_i)$ consists of factors $1/2$ for erasure-valued y_i, 1 for zero-valued y_i, and 0 for one-valued y_i.
$\longrightarrow$ if any of the channel outputs observed is 0, then the codeword is all-zero with probability 1. Vice versa, if any of the channel outputs observed is 1, then the codeword is all-one with probability 1.
Trick question: what if $y_i = 0$ and $y_j = 1$ for some i, j?

Symbol-wise decoding for the Single Parity-Check (SPC) code

For $X_1 = 0$, the sum over all codewords becomes a sum over all combinations of the symbols x_2 to x_N that sum to zero,

$$P_{X_1|Y_1\ldots Y_N}(0|y_1\ldots y_N) = \beta \sum_{\substack{x_1\ldots x_N\in\mathcal{C}\\ x_1=0}} \prod_{i=1}^{N} P_{X|Y}(x_i|y_i)$$

$$= \beta P_{X|Y}(0|y_1) \sum_{x_2+x_3+\ldots x_N=0} \prod_{i=2}^{N} P_{X|Y}(x_i|y_i),$$

and, similarly, for $X_1 = 1$, the sum over all codewords becomes a sum over all combinations of the symbols x_2 to x_N that sum to one,

$$P_{X_1|Y_1\ldots Y_N}(1|y_1\ldots y_N) = \beta P_{X|Y}(1|y_1) \sum_{x_2+x_3+\ldots x_N=1} \prod_{i=2}^{N} P_{X|Y}(x_i|y_i).$$

Example: $N = 3$

Let us consider the simple example of a length $N = 3$ single parity-check code,

$$\begin{aligned} P_{X_1|Y_1Y_2Y_3}(0|y_1y_2y_3) = \beta P_{X|Y}(0|y_1) \big[& P_{X|Y}(0|y_2)P_{X|Y}(0|y_3) \\ & + P_{X|Y}(1|y_2)P_{X|Y}(1|y_3) \big] , \end{aligned}$$

and

$$\begin{aligned} P_{X_1|Y_1Y_2Y_3}(1|y_1y_2y_3) = \beta P_{X|Y}(1|y_1) \big[& P_{X|Y}(0|y_2)P_{X|Y}(1|y_3) \\ & + P_{X|Y}(1|y_2)P_{X|Y}(0|y_3) \big] , \end{aligned}$$

Example: $N = 3$

Defining the vectors $p_i = [P_{X|Y}(0|y_i), P_{X|Y}(1|y_i)]$, we can write the previous expressions in vector form as

$$P_{X_1|Y_1Y_2Y_3}(0|y_1y_2y_3) = \beta p_{10}(p_2 \cdot p_3^T)$$

and

$$P_{X_1|Y_1Y_2Y_3}(1|y_1y_2y_3) = \beta p_{11}(p_2 \cdot (\mathbf{S}p_3)^T),$$

where **S** is the cyclic shift matrix, in this case

$$\mathbf{S} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$\longrightarrow$ This is a cyclic convolution of the a-posteriori probability distributions for y_2 and y_3.

Single Parity-Check (SPC) code

Generalisation

Extending to any dimension N, this observation remains valid: the output is the elementwise product of the a-posteriori distribution for y_1 with the cyclic convolutions of the a-posteriori probability distributions for y_2 to y_N.

This is also true for non-binary codes over primary fields GF(p). It is slightly different for codes over extension fields, because the convolution is no longer the cyclic convolution, but a component-wise cyclic convolution.

Cyclic convolutions

The most efficient way to take cyclic convolutions is taking the Discrete Fourier Transform, multiplying componentwise, then taking the inverse Discrete Fourier Transform.

Cyclic Convolutions and the Discrete Fourier Transform

For binary codes, we simply need a two-point Discrete Fourier Transform:

$$\begin{bmatrix} P_0 \\ P_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}$$

and the corresponding inverse transform is almost identical

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix}.$$

Example decoding for a Single Parity-Check (SPC) code

We received the observations $+0.3, -1.2, -0.1, 0.8$ from a BPSK modulated AWGN channel with $\sigma^2 = 1$ and want to estimate the first symbol.

Step 1 Convert the received symbols into a-posteriori probability distributions using $p_{i0} = P_{X|Y}(+1|y_i) = 1/(1 + e^{-2y_i/\sigma^2})$ etc. to yield

$$\begin{bmatrix} p_{10} & p_{20} & p_{30} & p_{40} \\ p_{11} & p_{21} & p_{31} & p_{41} \end{bmatrix} = \begin{bmatrix} 0.65 & 0.08 & 0.45 & 0.83 \\ 0.35 & 0.92 & 0.55 & 0.17 \end{bmatrix}$$

Step 2 Take the two point DFT of each column to yield

$$\begin{bmatrix} P_{10} & P_{20} & P_{30} & P_{40} \\ P_{11} & P_{21} & P_{31} & P_{41} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.3 & -0.84 & -0.1 & 0.66 \end{bmatrix}$$

Example decoding for a Single Parity-Check (SPC) code

Step 3 Multiply columns 2 to 4 elementwise to yield $[1, 0.016632]^T$

Step 4 Take the inverse two-point DFT of the above to obtain $[0.5083, 0.4917]^T$.

Step 5 Multiply by the a-posteriori probabilities for X_1 and renormalise, to yield

$$\begin{cases} P_{X_1|Y_1 \dots Y_4}(0|y_1 \dots y_4) & = \beta 0.5083 \times 0.65 \\ & = \beta 0.3304 = 0.6575 \\ P_{X_1|Y_1 \dots Y_4}(1|y_1 \dots y_4) & = \beta 0.4917 \times 0.35 \\ & = \beta 0.1721 = 0.3425 \end{cases}$$

Non-binary codes

Although our example was for a single parity-check code on a binary-input channel, the method extends to non-binary codes:

- For codes over base fields GF(p), take the p-point complex Discrete Fourier Transform (DFT) of each APP distribution, multiply componentwise, then take the inverse DFT to yield the cyclic convolution of all a-posteriori distributions.
- For codes over extension fields GF(2^m), the Walsh-Hadamard (WH) transform maps the component-wise binary cyclic convolution to a multiplication (just as the DFT for cyclic convolution.) The WH is a simple transform in the real domain that can be represented by matrices of the form

$$W = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{array}\right], W^{-1} = \frac{1}{4}W$$

  The Fast Hadamard Transform (FHT) computes it efficiently just like the Fast Fourier Transform (FFT) for the DFT.

Simplifying the binary case

The DFT method for the binary case boils down to:

1. Compute the DFT, i.e., the sum $P_0 = p_0 + p_1 = 1$ of the APPs and the difference $P_1 = p_0 - p_1$ for each observation
2. Compute the products $\prod_{i=2}^{N} P_{i0} = 1$ and $\prod_{i=2}^{N} P_{i1}$
3. Compute the inverse DFT as the sum

$$\frac{1}{2}\left(\prod_{i=2}^{N} P_{i0} + \prod_{i=2}^{N} P_{i1}\right) = \frac{1}{2}\left(1 + \prod_{i=2}^{N}(p_{i0} - p_{i1})\right)$$

   and the difference

$$\frac{1}{2}\left(\prod_{i=2}^{N} P_{i0} - \prod_{i=2}^{N} P_{i1}\right) = \frac{1}{2}\left(1 - \prod_{i=2}^{N}(p_{i0} - p_{i1})\right).$$

4. Multiply by p_{10} and p_{11} and normalise the result

Simplifying the binary case

This yields the overall mapping

$$p_{i0}^{o} = \frac{\frac{1}{2}\left(1 + \prod_{j=\backslash i}(p_{j0} - p_{j1})\right) p_{i0}}{\frac{1}{2}\left(1 + \prod_{j=\backslash i}(p_{j0} - p_{j1})\right) p_{i0} + \frac{1}{2}\left(1 - \prod_{j=\backslash i}(p_{j0} - p_{j1})\right) p_{i1}}$$

where we use the notation p_{i0}^{o} for the APP after decoding $p_{i0}^{o} = P_{X_i|Y_1\ldots Y_N}(0|y_1 \ldots y_N)$, and $\prod_{j=\backslash i}$ for a product over all but the i-th index.

Simplifying, we obtain

Decoding for the binary SPC

$$p_{i0}^{o} = \frac{\left(1 + \prod_{j=\backslash i}(p_{j0} - p_{j1})\right) p_{i0}}{1 + \prod_{j=1}^{N}(p_{j0} - p_{j1})}$$

and

$$p_{i1}^{o} = \frac{\left(1 - \prod_{j=\backslash i}(p_{j0} - p_{j1})\right) p_{i1}}{1 + \prod_{j=1}^{N}(p_{j0} - p_{j1})}$$

Binary decoding of SPC codes in the log domain

From the expressions on the previous slide, we know that the LLR we are after is

$$L_i^o = \log \frac{P_{X|Y_1 \dots Y_N}(0|y_1 \dots y_N)}{P_{X|Y_1 \dots Y_N}(1|y_1 \dots y_N)} = \log \frac{p_{i0}\left(1 + \prod_{j=\setminus i}(p_{j0} - p_{j1})\right)}{p_{i1}\left(1 - \prod_{j=\setminus i}(p_{j0} - p_{j1})\right)}$$

using Bayes' rule to convert between likelihoods and APPs in the case of uniform binary inputs. This can be re-written as

$$L_i^o - L(y_i) = \eta = \log \frac{1+\alpha}{1-\alpha} \tag{1}$$

where $\alpha = \prod_{j=\setminus i}(p_{j0} - p_{j1})$. Taking the exponent and re-arranging (1), we obtain $\frac{e^\eta - 1}{e^\eta + 1} = \alpha$. Furthermore, we can re-write α as

$$\alpha = \prod_{j=\setminus i} \frac{p_{j0} - p_{j1}}{p_{j0} + p_{j1}} = \prod_{j=\setminus i} \frac{p_{j0}/p_{j1} - 1}{p_{j0}/p_{j1} + 1} = \prod_{j=\setminus i} \frac{e^{L(y_j)} - 1}{e^{L(y_j)} + 1}.$$

Binary decoding of SPC codes in the log domain

Summarising the results of the previous slide, we obtained that

$$\frac{e^{L_i^o - L(y_i)} - 1}{e^{L_i^o - L(y_i)} + 1} = \prod_{j=\backslash i} \frac{e^{L(y_j)} - 1}{e^{L(y_j)} + 1}.$$

We now note that for any x, $\frac{e^x - 1}{e^x + 1} = \frac{e^{x/2} - e^{-x/2}}{e^{x/2} + e^{-x/2}} = \tanh \frac{x}{2}$, leading to

$$\tanh \frac{L_i^o - L(y_i)}{2} = \prod_{j=\backslash i} \tanh \frac{L(Y_j)}{2},$$

and finally

LLR Decoding Rule for a Single Parity-Check (SPC) Code

The output LLR for the i-the symbol is

$$L_i^o = L(y_i) + 2 \tanh^{-1} \prod_{j=\backslash i} \tanh \frac{L(y_j)}{2}.$$

Special Case: SPC over BEC

For the Binary Erasure Channel, it is is easy by inspecting the probability domain rules to deduct the following rule:

Decoding Rule for the BEC

For the *i*-th symbol,

- if p_{i0} is one (received a non-erased 0), decode $\hat{x}_i = 0$
- if p_{i0} is zero (received a non-erased 1), decode $\hat{x}_i = 1$
- if $p_{i0} = 1/2$, and all other symbols were received non-erased, $\hat{x}_i$ equals the parity of the remaining symbols
- if $p_{i0} = 1/2$ and any other symbol has also been received as an erasure, we cannot decode (flip a coin to guess $\hat{x}_i$) as the APP is $p^o_{i0} = p^o_{i1} = 1/2$.

Summary for the two curious codes

Code/Channel	General binary-input	Binary Erasure Channel
Repetition Code	$L_i^o = \sum_j L(y_j)$	Decode to the value of *any* symbol received non-erased
Single Parity-Check (SPC) code	$L_i^o = L(y_i) + 2\tanh^{-1} \prod_{j=\backslash i} \tanh \frac{L(y_j)}{2}$	Decode if x_i received non-erased OR if *all* other symbols received non-erased

General linear codes: iterative decoding

$$\mathbf{H} = \begin{array}{cc} \begin{array}{cccccccccccc} & & & & \text{Repetition code} & & & & & & & \end{array} & \\ \left[\begin{array}{cccccccccccc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{array}\right] & \begin{array}{l} \\ \\ \\ \\ \text{SPC code} \\ \\ \end{array} \end{array}$$

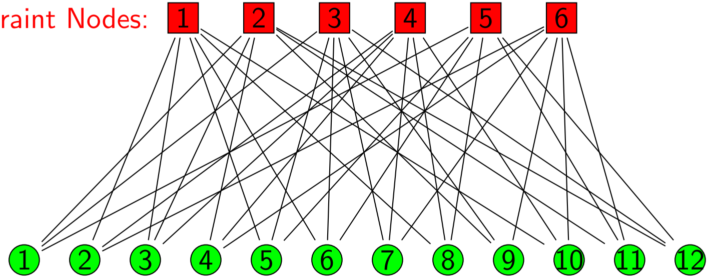
- *Locally*, every parity-check equation in a linear code is a Single Parity-Check (SPC) code (use SPC decoding rule to get incremental improvement)
- *Locally* again, the LLRs contributed by each parity-check equation for one code symbol at position i can be seen as a repetition code for X_i (use repetition code decoding rule to get incremental improvement)

$\longrightarrow$ iterative decoding

Factor Graphs

$$
\mathbf{H} = \begin{array}{c}
\begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\
\left[\begin{array}{cccccccccccc}
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0
\end{array}\right]
\end{array}
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
$$

Constraint Nodes:



Variable Nodes

Extrinsic Decoding

When applying decoding rules in an iterative setup, it is essential *not* to use the observed symbol in the decoding operation for itself (positive feedback, self-fulfilling prophecy), hence:

Extrinsic decoding rules for SPC and repetition codes

For repetition codes, exclude the observation corresponding to the current symbol X_i, i.e.,

$$L_i^{ex} = \sum_{j=\backslash i} L(y_j).$$

For SPC codes, omit the LLR of the current symbol X_i, i.e.,

$$L_i^{ex} = 2\tanh^{-1} \prod_{j=\backslash i} \tanh \frac{L(y_j)}{2}.$$

Note that for the BEC, it can be shown that extrinsic decoding makes no difference and hence one can iterate using the plain BEC decoding rules.

Message Passing

1. Initialise all graph messages to uniform distributions (zero LLRs)
2. At each iteration k,
   1. For each variable node, apply the repetition code rule to update the LLR/distribution on each outgoing edge based on the incoming LLRs/distributions and on the channel APP,
   $$L^{ex,k}_{v \to c,i} = L_{ch} + \sum_{j = \setminus i} L^{ex,k-1}_{c \to v,j}.$$
   2. For each constraint node, apply the SPC code rule to update the LLR/distribution on each outgoing edge based on the incoming LLRs/distributions,
   $$L^{ex,k}_{c \to v,i} = 2 \tanh^{-1} \prod_{j = \setminus i} \tanh(L^{ex,k}_{v \to c,j}/2).$$
3. Last iteration: for each variable node, compute the (non-extrinsic) APP for each variable node,
$$L^{o} = L_{ch} + \sum_{j} L^{ex}_{c \to v,j}.$$

C-Code: binary iterative decoding

```
while (1) {

  /* VARIABLE NODE OPERATIONS */
  cumdeg = 0;
  for (j = 0; j < H.N ; j++) {
    x = 0.0;
    for (k = 0 ; k < H.vdeg[j] ; k++)
      x += d.msg[H.intrlv[cumdeg+k]];
      app[j] = x+chmsg[j];
      /* app contains the a-posteriori LLR */
      for (k = 0 ; k < H.vdeg[j] ; k++)
        d.msg[H.intrlv[cumdeg+k]] = app[j]
- d.msg[H.intrlv[cumdeg+k]]; /* extr. LLR */
    }
    cumdeg += H.vdeg[j];
  }

  /* STOPPING RULE */
  ...
  if (checksum) {
    stopflag = 0;
    break;
  }
```

```
  /* CONSTRAINT NODE OPERATIONS */

  cumdeg = 0;
  for (j = 0; j < H.M ; j++) {
    app[j] = 1.0;
    for (k = 0 ; k < H.cdeg[j] ; k++)
      app[j] *= tanh(d.msg[cumdeg+k]/2.0);
    for (k = 0 ; k < H.cdeg[j] ; k++)
      d.msg[cumdeg+k] = 2.0*arctanh(app[j]
 / d.msg[cumdeg+k]); /* extrinsic LLR */
    }
    cumdeg += H.cdeg[j];
  }

  it++;
} /* while (1) */
```

C-Code: non-binary iterative decoding

```
while (1) {

  /* VARIABLE NODE OPERATIONS */
  cumdeg = 0;
  for (j = 0; j < H.N ; j++) {
    for (m = 0 ; m < H.q ; m++) {
      x = 1.0;
      for (k = 0 ; k < H.vdeg[j] && it != 0 ; k++)
        x *= d.msg[H.oi[(cumdeg+k)*H.q+m]];
      app[j*H.q+m] = x*chmsg[j*H.q+m];
      /* app contains the a-posteriori prob */
      for (k = 0 ; k < H.vdeg[j] ; k++)
d.msg[H.oi[(cumdeg+k)*H.q+m]] = app[j*H.q+m]
/ d.msg[H.oi[(cumdeg+k)*H.q+m]]; /* extr. prob */
    }
    cumdeg += H.vdeg[j];
  }
  renorm_msg(d.msg, d.L, d.q);

  /* STOPPING RULE */
  ...
  if (checksum) {
    stopflag = 0;
    break;
  }
```

```
  /* CHECK NODE OPERATIONS */

  /* Hadamard transform of messages */
  hadamard_msg(d.msg, d.L, d.q, H.log2q);

  /* compute check node rule */
  cumdeg = 0;
  for (j = 0; j < H.M ; j++) {
    for (m = 0 ; m < H.q ; m++) {
      app[j*H.q+m] = 1.0;
      for (k = 0 ; k < H.cdeg[j] ; k++)
app[j*H.q+m] *= d.msg[(cumdeg+k)*H.q+m];
for (k = 0 ; k < H.cdeg[j] ; k++)
  d.msg[(cumdeg+k)*H.q+m] = app[j*H.q+m]
 / d.msg[(cumdeg+k)*H.q+m]; /* extrinsic prob */
    }
    cumdeg += H.cdeg[j];
  }

  /* inverse Hadamard transform of messages */
  hadamard_msg(d.msg, d.L, d.q, H.log2q);
  for (k = 0 ; k < d.q*d.L ; k++)
    d.msg[k] /= d.q;

  it++;
} /* while (1) */
```

Low-Density Parity-Check (LDPC) Codes

Success of Iterative Decoding

Iterative decoding will only converge and have a low complexity if the parity-check matrix has low density, i.e., if the associate factor graph is sparse.

- Think of the BEC: repeat code wants as large an N as possible (any non-erased observation enables decoding), wheras SPC code wants its N to be as small as possible (any erased observation blocks decoding).
- Column weights and row weights are linked (total number of ones in matrix).
- Number of decoder operations per iteration proportional to number of edges (ones in $\mathbf{H}$ matrix)

Regular/irregular LDPC Codes

The *degree* of a node in a graph is the number of edges connected to it.

Regular LDPC codes

An (d_c, d_v) regular code is a code for which every row has d_c ones and every column has d_v ones, or, equivalently, every constraint node in the associated factor graph has degree d_c, while every variable node has degree d_v.

Irregular LDPC codes

In an irregular LDPC code, the proportion of nodes of certain degrees is determined by degree polynomials.

Degree polynomials

Degree polynomials of irregular LDPC codes come in two forms:

- node perspective polynomials specify the proportion of nodes of each degree,i.e.,

$$L(x) = \sum_{i=1}^{d_v^{\max}} L_i x^i, R(x) = \sum_{i=1}^{d_c^{\max}} R_i x^i$$

  where L_i is the proportion of variable nodes of degree i, R_i is the proportion of constraint nodes of degree i, and x is a meaningless auxiliary variable used for notation only.

- edge perspective polynomials specify the proportion of edges connected to nodes of each degree, i.e.,

$$\lambda(x) = \sum_{i=1}^{d_v^{\max}} \lambda_i x^{i-1}, \rho(x) = \sum_{i=1}^{d_c^{\max}} \rho_i x^{i-1},$$

  where λ_i is the proportion of edges connected to variable nodes of degree i, while ρ_i is the proportion of edges connected to constraint nodes of degree i.

Rate of an LDPC Code

The rate of a regular (d_c, d_v) LDPC code can be deduced directly from its degrees, independently of its block length:

- Let us assume that it has block length N and dimension K
- Its parity-check matrix has $(N - K)$ rows and N columns
- The total number of ones in the parity-check matrix is then $(N - K)d_c = Nd_v$ and, dividing by N and rearranging, we obtain

$$R = 1 - d_v/d_c$$

For irregular codes, it is easy to show that

- the average degrees are

$$\bar{d}_v = L'(1) = 1/\left(\int_0^1 \lambda(x)dx\right) \text{ and } \bar{d}_c = R'(1) = 1/\left(\int_0^1 \rho(x)dx\right)$$

  where $L'()$ and $R'()$ are the derivatives of the corresponding degree polynomials.
- the rate is $$R = 1 - \frac{L'(1)}{R'(1)} = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}.$$

LDPC Codes: Code Construction

- Specifying degree polynomials only specifies a class of parity-check matrices, not a specific matrix or code
- *Random construction:* pick a random matrix that fits the degree polynomials. Equivalently, pick a random interleaver to connect the edge terminals on the variable nodes to the edge terminals on the constraint nodes.
- *Improved random construction:* same as above but excluding bad cases (parallel connections between nodes, short cycles of length 4, 6, etc.). The *girth* of the graph is the length of its shortest cycle, and keeping cycles long will minimise the effect of positive feedback in the iterative decoder. Standard method: progressive edge growth (PEG).
- *Geometric/algebraic construction:* systematic construction of (typically regular) codes using geometry, algebra, etc. Advantage: construction reproducible without storing the actual (large) matrix. Disadvantage: performance for long block lengths tends to be worse than random codes.