AKA Gallager Codes

Low Density Parity Check Codes

Aditya

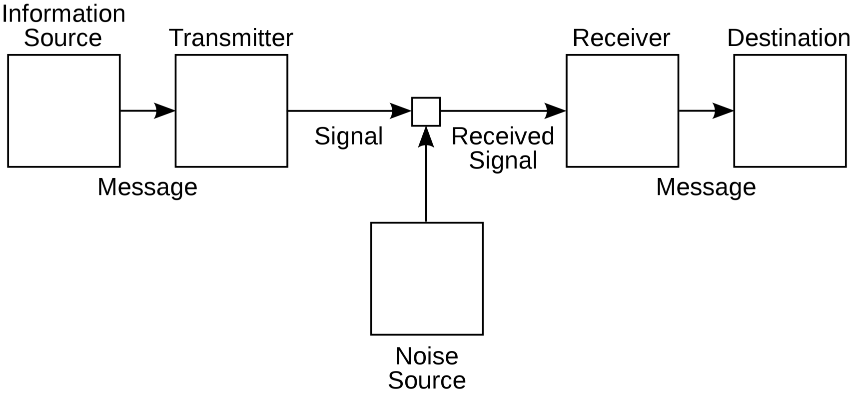
Outline

Section 1

Formalizing Error Correction Codes

The Model

- Transmitter AKA modulator does bits $\mapsto$ signals.
- In our model, noise source adds Gaussian noise $n(t)$ that is independent from symbol to symbol.



Modulation

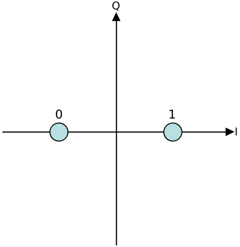
- Remember, the goal is to map bits (0s and 1s) to a signal.
- We will use binary phase shift keying (BPSK), which works by changing (*modulating*) the phase of a basis function. Then, $0 \mapsto s_0(t)$ and $1 \mapsto s_1(t)$:

$$s_0(t) = \sqrt{\frac{2E}{T}} \sin\left(2\pi f t + \frac{\pi}{2}\right)$$
$$s_1(t) = \sqrt{\frac{2E}{T}} \sin\left(2\pi f t - \frac{\pi}{2}\right)$$

- Looks hard, but phasors can help!

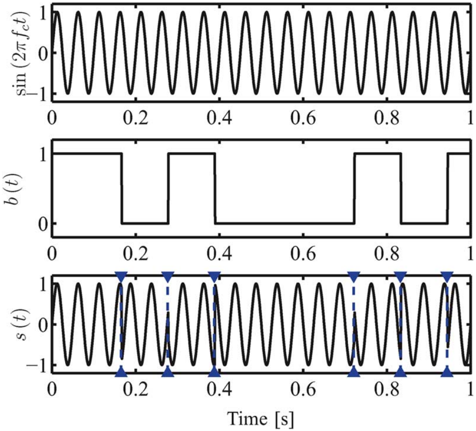
Constellations

- Constellation diagram (very similar to phasor diagram) – in $\mathbb{C}$, the argument gives the phase shift, and the norm gives the amplitude of a signal.



- So, BPSK maps $0 \mapsto +1$ and $1 \mapsto -1$. Note that this is a mapping from $\mathbb{F}_2 \mapsto \mathbb{R}$.

BPSK in time domain



Through the channel

We add in white Gaussian noise (AWGN).

$$y(t) = x(t) + n(t)$$

where $x(t)$ is the input signal, and $n(t)$ is a Gaussian process, and independent for each symbol.

Demodulate!

$$\rho = \int_0^T y(t)\sqrt{\frac{2E}{T}}\sin\left(2\pi ft + \frac{\pi}{2}\right)dt$$

- Important: $\rho \in \mathbb{R}$.
- Why does this work? There's a little bit of analog signal processing that's not too relevant...in essence, the process involves re-multiplying by the carrier signal, then using a low pass filter to pick out the data.

What do we do with ρ?

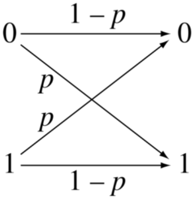
Remember, we need to map back from $\mathbb{R} \mapsto \mathbb{F}_2$.
Hard decision decoding: Threshold at 0 to get the output bit:

$$b = \begin{cases} 0, & \rho > 0 \\ 1, & \rho \leq 0 \end{cases}$$

Soft decision decoding: We'll talk about it soon!

Binary symmetric channels

- It can be shown that BPSK over AWGN is a BSC.



How to compute bit-flip probability p?

The bit-error rate p is given by:

$$\text{BER} = P(t = +1) \cdot P(n \leq -1) + P(t = -1) \cdot P(n \geq +1)$$

Assuming an even mix of 0s and 1s,

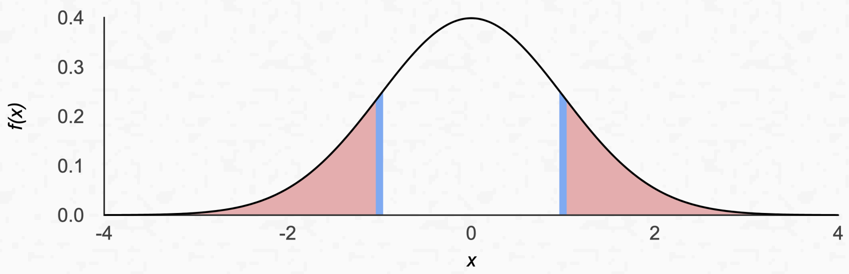
$$\text{BER} = \frac{1}{2} P(n \leq -1) + \frac{1}{2} P(n \geq +1)$$

Recall, the noise is a Gaussian distribution with variance σ.

After some statistics...

The transition probability of our model is:

$$p = \mathrm{BER} = Q\left(\frac{1}{\sigma}\right)$$



$$Q(x) := \frac{1}{2}\mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

One last thing: power!

Signal-to-noise ratio:

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} \left(\frac{P_{\mathrm{signal}}}{P_{\mathrm{noise}}} \right)$$

In discrete time,

$$\mathrm{SNR} = \frac{E_s}{\sigma^2} = \frac{E_s}{\frac{N_0}{2}} = \frac{2E_s}{N_0}$$

where E_s is the energy per symbol, $N_0/2$ is the power spectral density (variance) of the noise signal.

SNR for BPSK

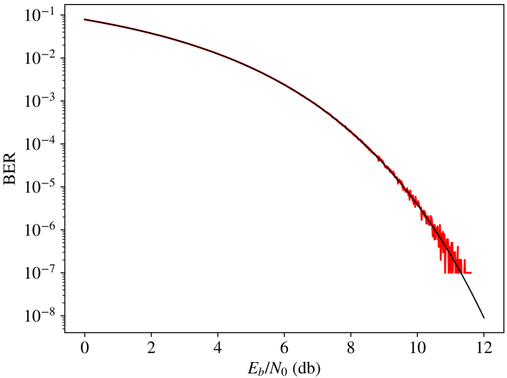
$$E_s = \frac{(-1)^2 + 1^2}{2} = 1 \implies \mathrm{SNR} = \frac{1}{\sigma^2}$$

We get a nice relation between SNR and BER for our model:

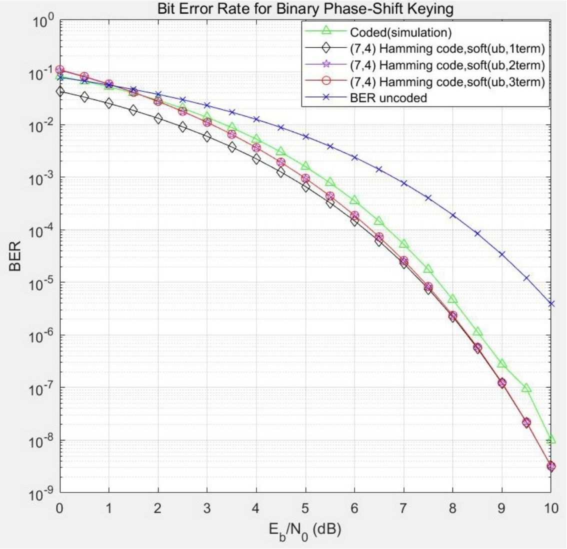
$$\mathrm{BER} = Q\left(\sqrt{\mathrm{SNR}}\right)$$

SNR vs. BER

Red line is a Monte-Carlo simulation that counts bit errors for AWGN ($\sigma = 1$).We usually use E_b/N_0 (SNR per bit) instead of $2E_s/N_0$ (SNR) in these graphs.

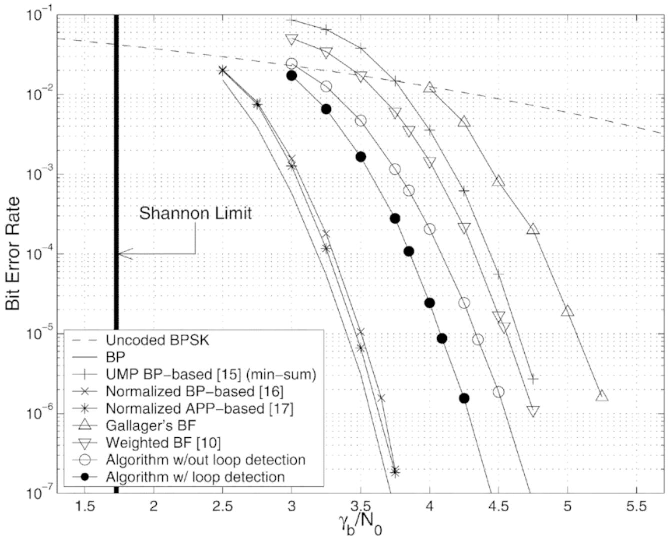


What do error correction codes do?



Shannon Limit and Capacity-Approaching Codes

$$E_b = E_s/R = nE_s/k$$



Section 2

Simple Soft Decision Decoders

$n = 3$ Repetition Code

What's the easiest way to make sure someone understands *exactly* what you're saying?

Repeat yourself (say it three times)!

Encoder

Note that the rate of this code is $k/n = 1/3$.

m	c	$\vec{s}$
0	000	`[+1, +1, +1]`
1	111	`[-1, -1, -1]`

Hard decision decoder

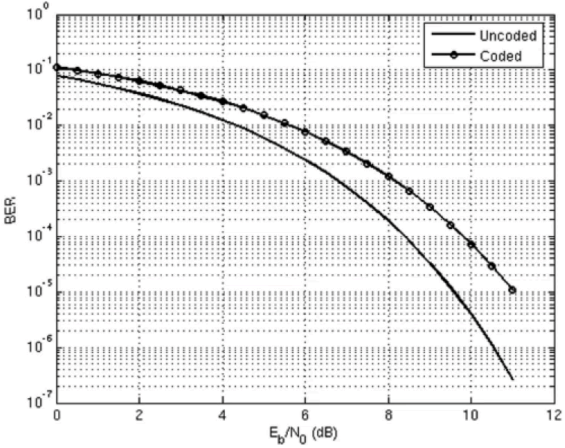
The output from the demodulator is some vector of real numbers, say $\vec{r} = [r_0, r_1, r_2]$. Then, hard decision decode this to $\vec{b}$ by thresholding at zero. Finally, use a majority function:

$\vec{b}$	$\hat{c}$
000	000
001	000
010	000
100	000
011	111
101	111
110	111
111	111

So, we're happy with ourselves

- Not so fast – let's analyze this code within the formal framework we laid out earlier.
$$\frac{E_b}{N_0} = \frac{E_s/\sigma^2}{2R} = \frac{3}{2\sigma^2}$$
- The probability of a bit-flip is then:
$$\implies p = Q\left(\sqrt{\frac{2E_b}{3N_0}}\right)$$
- The overall probability of an error is $\text{BER} = 3p^2(1-p) + p^3$.

Plotting the hard decision decoder for $n = 3$ repetition code



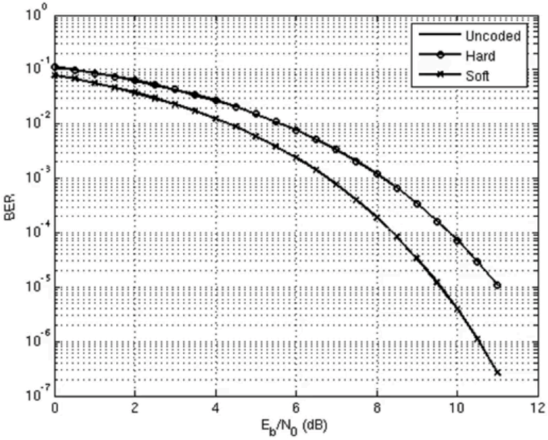
Soft decision decoding

- The received real vector $\vec{r}$ can be analyzed in a real vector space.
- Compare the correlation of $\vec{r}$ with the codewords, and pick the output symbol based on that. If:

$$\vec{r} \cdot \begin{bmatrix} +1 & +1 & +1 \end{bmatrix} > \vec{r} \cdot \begin{bmatrix} -1 & -1 & -1 \end{bmatrix}$$

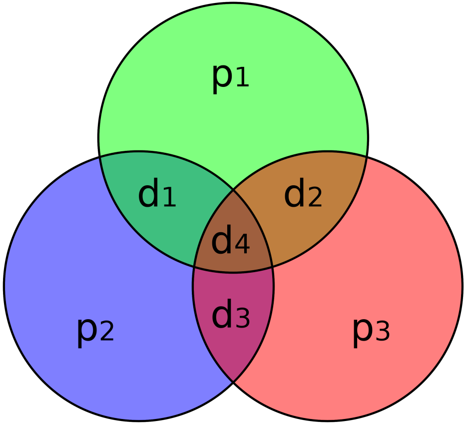
  $\hat{c} = 000$ else $\hat{c} = 111$.
- More simply, check $r_0 + r_1 + r_2 > 0$.
- This is an optimal maximum likelihood decoder.

BER vs SNR per bit for optimal decoding of repetition code



(7, 4) Hamming Code

Recall from Anakin's introductory meeting on codes the (7, 4) Hamming code.



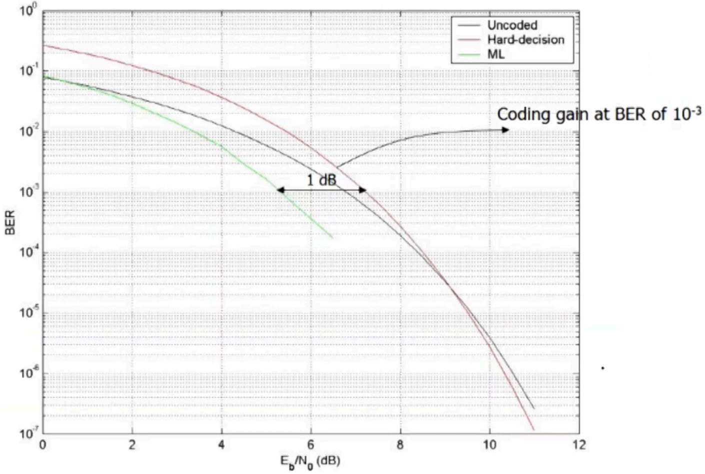
Hard decision decoder

- *After* the hard decision thresholding of the received vector $\vec{r}$ around 0 to get $\vec{b}$,
- Correct to the codeword at the closest Hamming distance from $\vec{b}$.
- This is the *minimum distance decoder* for the Hamming code.

Soft decision decoder

- Find the closest codeword to $\vec{r}$ in *Euclidean* distance.
- That is, in the vector space $\mathbb{R}^n$.
- Clearly, this is (much) more complex, and becomes hard to implement as k increases for a Hamming code.
- This is the *maximum likelihood decoder* for the Hamming code.

BER vs SNR per bit for Hamming (7,4) Decoders



SISO Decoding

- There is another kind of decoder, the soft-in soft-out decoder.
- We start with implementing it for the repetition code (this is really easy and just for demonstrating the technique).

Belief

- The output of the SISO decoder is a real vector $\vec{L} = [L_0 \ L_1 \ L_2]$, where each L_i indicates the strength of the “belief” that bit c_i of the codeword is (say) 0.
- What does this mean? Imagine you received the vector $[3.2, \ 4.3, \ 2.4]$.
- This indicates that it’s very likely, in each case, that the transmitted symbol was $+1$.

Tell me more about your beliefs

- What about $\vec{r} = [0.02, \ -3.2, \ -0.6]$?
- A hard-decision decoder would turn this into $[1, \ -1, \ -1]$.
- However, for a SISO decoder and a repetition code, you *know* that all the bits should be the same.
- How sure are you about 0.02?

Formalizing the intuition

The probabilities below are of interest (Bayes' rule):

$$P(c_0 = 0|r_0) = \frac{f(r_0|c_0 = 0)P(c_0 = 0)}{f(r_0)}$$

$$P(c_0 = 1|r_0) = \frac{f(r_0|c_0 = 1)P(c_0 = 1)}{f(r_0)}$$

It is natural to divide these quantities:

$$\frac{P(c_0 = 0|r_0)}{P(c_0 = 1|r_0)} = \frac{f(r_0|c_0 = 0)}{f(r_0|c_0 = 1)}$$

Intrinsic log likelihood ratios

Recall that the noise is normally distributed, so $f(r_0|c_0 = 0) = 1 + N(0, \sigma^2)$ and $f(r_0|c_0 = 1) = -1 + N(0, \sigma^2)$. Plugging in the Gaussian PDF and simplifying gives

$$\frac{P(c_0 = 0|r_0)}{P(c_0 = 1|r_0)} = \exp \frac{2r_0}{\sigma^2}$$

So, the *intrinsic log likelihood ratio* of r_0 is:

$$l_0 = \log \frac{P(c_0 = 0|r_0)}{P(c_0 = 1|r_0)} = \frac{2r_0}{\sigma^2}$$

This is general for any intrinsic LLR in BPSK/AWGN. (Typically, we ignore the constant factor here, since it's merely a constant scaling of our belief.)

Output log likelihood ratios

We still want to get L_i, which is a belief *in the context* of the other elements of the received vector $\vec{r}$. Formally, we want:

$$L_i = \log \frac{P(c_i = 0 | r_0, r_1, r_2)}{P(c_i = 1 | r_0, r_1, r_2)}$$

Skipping the Bayes' rule transformation, we see that:

$$L_i = \log \frac{f(r_0, r_1, r_2 | c_0 = 0)}{f(r_0, r_1, r_2 | c_0 = 1)}$$

Since this is an AWGN channel, each normal distribution in this joint PDF is independent, so, after inserting a product of similar distributions as in the intrinsic case, we simply get:

$$L_i = \frac{2}{\sigma^2}(r_0 + r_1 + r_2)$$

SISO Decoding a Repetition Code

Thus, after adjusting for the scaling factors, the SISO decoder output is given by

$$L_0 = \underbrace{r_0}_{\text{intrinsic}} + \underbrace{r_1 + r_2}_{\text{extrinsic}}$$

The “extrinsic” is really saying “what do r_1 and r_2 tell me about r_1?”
In our example ($[0.02, \; -3.2, \; -0.6]$), this would result in: $[-3.78, -3.78, -3.78]$.

A more interesting SISO Decoder: SPC Codes

- For a message m, XOR all the bits, and tack on the parity bit at the end. This is codeword c.
- This is the single parity check code.
- Consider, the (3, 2) SPC code:

m	c
00	000
01	011
10	101
11	110

- Let's design a SISO decoder, whose input is $\vec{r}$, and output is a 3-dimensional vector $\vec{L}$ of log likelihood ratios that corresponds to $\vec{r}$.

Extrinsic information

- It's clear what r_0 says about c_0: it's just the intrinsic belief.
- What do r_1 and r_2 say about c_0, though?
- Formally, we want:

$$l_{ext,0} = \log \frac{P(c_0 = 0|r_1, r_2)}{P(c_0 = 1|r_1, r_2)}$$

- We know $c_0 = c_1 \oplus c_2$. So,

$$P(c_0 = 0|r_1, r_2) = p_2 p_3 + (1 - p_2)(1 - p_3)$$

where

$$p_2 = \log \frac{P(c_2 = 0|r_2)}{P(c_2 = 1|r_2)} \quad p_3 = \log \frac{P(c_3 = 0|r_3)}{P(c_3 = 1|r_3)}$$

After some boring algebra...

We get that the relation $c_0 = c_1 \oplus c_2$ in the likelihood domain is

$$\tanh \frac{l_{ext,0}}{2} = \tanh \frac{l_1}{2} \cdot \tanh \frac{l_2}{2}$$

Breaking this up into the sign and the absolute values with logarithms,
$\operatorname{sgn} l_{ext,0} = \operatorname{sgn} l_1 \operatorname{sgn} l_2$

$$\log\left(\tanh \frac{|l_{ext,0}|}{2}\right) = \log\left(\tanh \frac{|l_1|}{2}\right) + \log\left(\tanh \frac{|l_2|}{2}\right)$$

Define $f(x) := \log \tanh |x|/2$. Then, $f(x) = f^{-1}(x)$. So,

$$|l_{ext,0}| = f(f(l_1) + f(l_2))$$

SISO Decoder for SPC Codes

$$L_0 = l_0 + l_{ext,0}$$

where

$$l_0 = \frac{2}{\sigma^2} r_0$$

and

$$|l_{ext,0}| = f(f(l_1) + f(l_2))$$

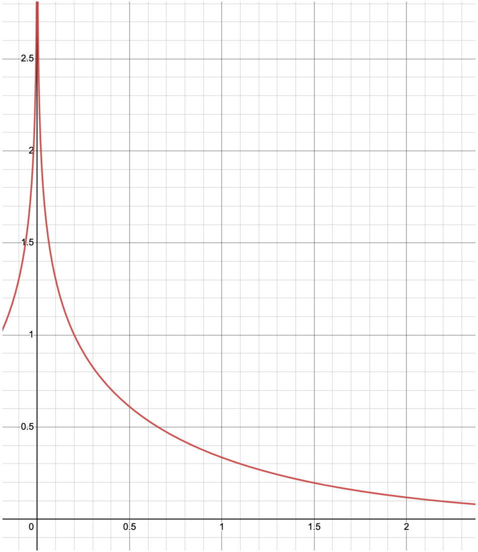
$$\operatorname{sgn} l_{ext,0} = \operatorname{sgn} l_1 \operatorname{sgn} l_2$$

where

$$f(x) := \log \tanh \frac{|x|}{2}$$

Computing $f(x)$ is hard

Approximate it!



Min-sum approximation

Small values dominate, so $f(|l_1|) + f(|l_2|) = f(\min(|l_1|, |l_2|))$. Translating back to our original formula,

$$|l_{ext,0}| = f(f(l_1) + f(l_2)) = f(f(\min(|l_1|, |l_2|))) = \min(|l_1|, |l_2|)$$

Since f is its own inverse.

SISO Decoder for General (n, n-1) SPC Code

Generalizes very naturally:

$$l_0 = \frac{2}{\sigma^2} r_0$$

and

$$l_{ext,0} = (\operatorname{sgn}(l_1) \operatorname{sgn}(l_2) \cdots \operatorname{sgn}(l_{n-1})) \min(|l_1|, |l_2|, \ldots, |l_{n-1}|)$$

...and so on for each L_i. Low-hanging optimizations here for both the sign and the minimum operations.

Section 3

Formalizing Linear Block Codes

Introduction

- From Wikipedia: “A linear code of length n and dimension k is a linear subspace C with dimension k of the vector space $\mathbb{F}_q^n$ where $\mathbb{F}_q$ is the finite field with q elements.”
- More simply, a linear block code takes an input vector of bits $\vec{m}$, and produces $\vec{c} = [\vec{m}\ \vec{p}]$, where $\vec{p}$ is the *parity check vector.*
- $\vec{m}$ is of dimension (length) k, $\vec{p}$ is of dimension p, and $\vec{c}$ is of dimension $n = k + p$.
- The elements of $\vec{p}$ are computed by XORing (adding modulo 2) certain bits of $\vec{m}$.

Example of simple (6, 3) linear block code

Parity computation is given by:

$$\begin{aligned} p_0 &= m_0 \oplus m_1 \\ p_1 &= m_1 \oplus m_2 \\ p_2 &= m_2 \oplus m_0 \end{aligned}$$

Clearly, the rate is $R = 1/2$.

Generator matrices

Clearly,

$$\begin{bmatrix} p_0 & p_1 & p_2 \end{bmatrix} = \begin{bmatrix} m_0 & m_1 & m_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

To get the full systematic codeword, tack on I_3:

$$\begin{bmatrix} m_0 & m_1 & m_2 & p_0 & p_1 & p_2 \end{bmatrix} = \begin{bmatrix} m_0 & m_1 & m_2 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}}_{G}$$

This matrix is known as the generator matrix for the code: $G = [I_k \ P]$. It has rank k, and its rows form the basis for the code space.

Parity check matrix

- Given by $H = [P^T \ I_{n-k}]$, a $(n-k) \times n$ matrix.

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}}_{H} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ p_0 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- In general, given a codeword, $Hc^T = 0$.

Exercise

Construct the generator matrix G and parity check matrix H for the $n = 3$ repetition code.
Bonus: do the same for the (7, 4) Hamming code.

Solution

$$G = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Section 4

Low Density Parity Check Codes

Some of the keywords should now make sense

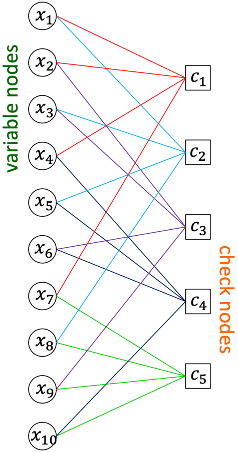
- LDPC codes are linear block codes with a very sparse parity check matrix H.
- That is, `popcount`(`H`) $<< n(n-k)$.

Tanner Graphs and Parity Check Matrices

Important: any one row of H that is, each check node corresponds to a single parity check code.

$$\mathbf{H} = \begin{array}{ccccccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & c_1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & c_2 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & c_3 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & c_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & c_5 \end{array}$$

(a) Parity-check matrix



(b) Tanner graph

Code Generation and Encoding

- Isn't terribly interesting, and we may come back to it later.
- Fundamentally the same idea as encoding any linear block code: a matrix multiplication (alternately, using the parity check matrix to figure out which bits to XOR).
- To optimize code performance, encoding complexity, memory footprint, a "base matrix" is carefully selected, then expanded in a certain way using circulant matrices to get the parity check matrix.
- The really interesting part of LDPC is the decoding algorithm.

LDPC Decoding

- SISO
- Iterative, belief propagation algorithm
- Uses the min-sum approximation from earlier
- For SISO decoding, recall that we want

$$L_i = \log \frac{P(c_i = 0|\vec{r})}{P(c_i = 1|\vec{r})}$$

that indicates the strength of the "belief" that bit c_i of the codeword is (say) 0.

Plan: use the Tanner graph

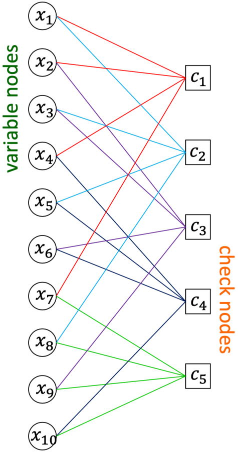
- Variable nodes (LHS) are connected to check nodes (RHS).
- Pass extrinsic information through the edges of the graph, so all the nodes "work together", adding their knowledge.
- Four steps of the decoding algorithm:
  1. Initialization
  2. Check-node processing
  3. Variable-node processing
  4. If syndrome is not zero or maximum iterations not reached, GOTO 2.

Visualization in the Tanner Graph

Initialize all the variable nodes with their channel (intrinsic) LLR l_i.

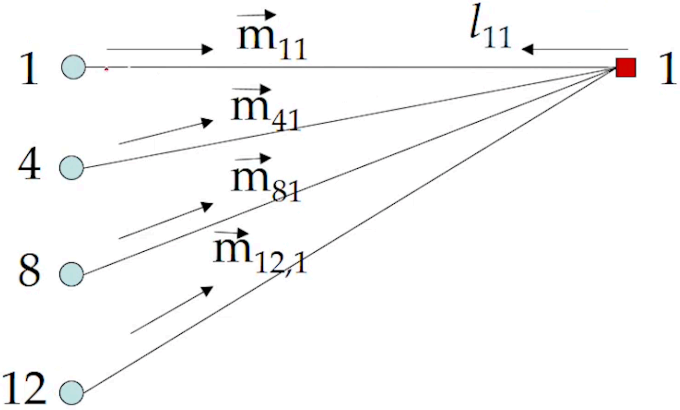
$$\mathbf{H} = \begin{array}{ccccccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & c_1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & c_2 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & c_3 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & c_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & c_5 \end{array}$$

(a) Parity-check matrix



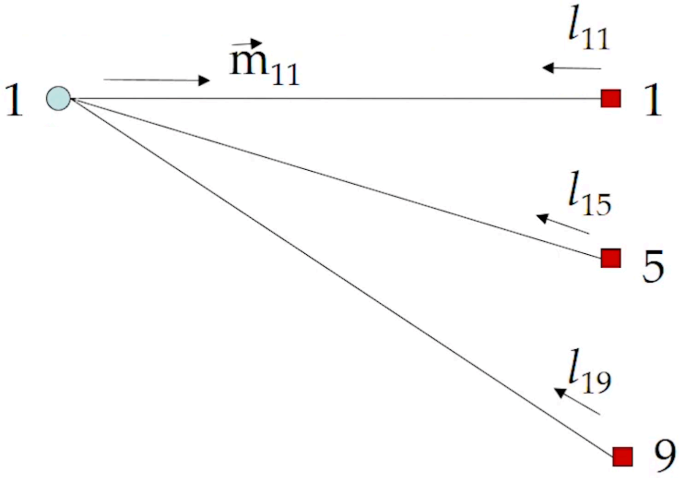
(b) Tanner graph

Check-node processing



This is an SPC! Check node (1) (say β_1) will do a SISO SPC decoding.

Variable-node processing



Each check node returns the *extrinsic* information from the SPC computation for each variable node (say α_i). This forms a repetition code!

Some properties

- More iterations is better
- Using the min-sum approximation causes a degradation in error-rate performance, but makes SISO SPC check node decoders very simple.
- Small cycles in the Tanner graph (low girth) can ruin performance for iterative decoding.
- Characterizing performance of LDPC codes requires “density evolution” analysis.

Thanks for coming!

From `https://www.inference.org.uk/mackay/codes/gifs/`