STANFORD

Lecture 10

Constraints & LDPC Codes

February 13, 2024

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

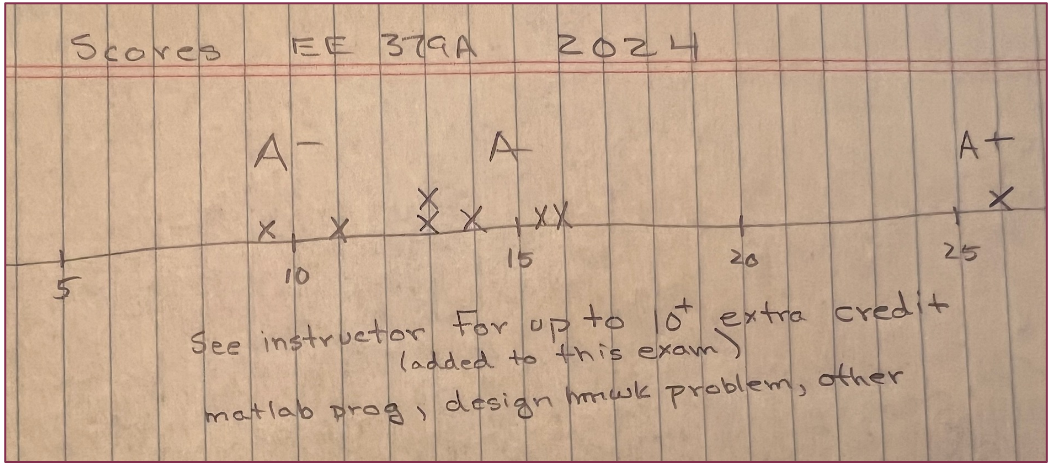
Instructor EE379A – Winter 2024

Announcements & Agenda

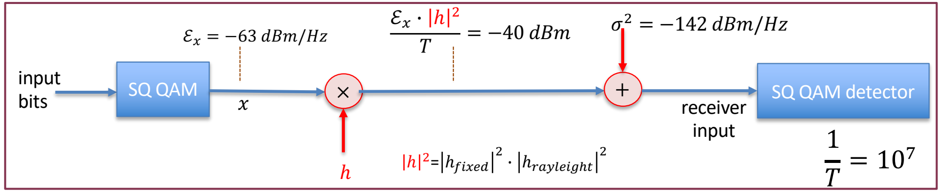
- Announcements

- Today
  - Midterm Review
  - Turbo-Code Completion
  - Soft Information from constraints
  - LDPC Codes
  - Hard/Soft concatenation – Reed Solomon outer

- Problem Set 5 = PS5 due Wednesday February 21
  1. 8.12 Turbo Design and Coding
  2. 8.13 Constraints and BICM
  3. 8.14 LDPC Use
  4. 8.15 Subsymbol- vs Symbol-Level Deterministic Interleaving
  5. 8.16 Wireless Hard-Soft Interleaving Challenge



Solutions



1. QAM Design

a. **Attenuation** -63-(-40-70) = **47 dB**; **SNR** =-110-(-142+3) = **29dB**

b. $P_e = 4\left(1-\frac{1}{16}\right) Q\left(\sqrt{\frac{3 \cdot 10^{2.9}}{255}}\right)$ = **.0042**

c. $C = 10^7 \cdot log_2(1+10^{2.9})$ = **96Mbps**

d. SNRnew = 10*log10(255/3*(qfuncinv(1e-6/3.75))^2) = 33.3 +.2, $\gamma = 4.5\ or\ 4.8$ **dB**

  a. **dfree=6 and G=[17 13] from tables.**

e. $SNR_{min} = 0\ (-\infty dB)$ and $SNR_{max} = +\infty dB$ (also +26 and smaller)

f. $\langle P_e \rangle = \frac{1}{2}\left[1 - \sqrt{\frac{\kappa SNR}{\kappa SNR+1}}\right]$ = **.0248**

g. SNRnew = 10*log10(63/3*(qfuncinv(1e-6/3.5))^2) = 27.3 dB

- This is 1.7 dB below average at 29 dB ; $10^{-1.7} = .6761$
- Coding of course will help.

$$P_{out} = \int_0^{.6761} \frac{1}{2} \cdot e^{-x/2} \cdot dx = 1 - e^{-.6761/2} = .2868$$

$$\boxed{\begin{aligned} g &\geq -1.7 - 6.3 = -8 \text{ dB} = .1585 \\ P_{out} &= .1 - e^{-.1585/2} = 0762 \text{ or } 7.6\% \\ g &\geq -14.6 \text{ dB}(.0316) \\ P_{out} &= 1 - e^{10^{-.0316/2}} = .0157 \text{ or } 1.6\ \% \end{aligned}}$$

2. Bridge SOVA / APP

a. $2^6 =$ **64 paths** ; $2^\nu =$ **4 survivors**

b. 2^6 x 2^{12} **table entries**

c. **BSC for 2 successive bits**

$$p_{\boldsymbol{v}'=00/\boldsymbol{v}} = \begin{cases} (1-p)^2 & \boldsymbol{v} = 00 \\ (1-p)\cdot p & \boldsymbol{v} = 01 \\ p \cdot (1-p) & \boldsymbol{v} = 10 \\ p^2 & \boldsymbol{v} = 11 \end{cases}$$

d. **No,** encoder has memory

e. **4 survivors**

$$\begin{aligned} p_{\boldsymbol{u}=000011/\boldsymbol{v}'} &= p^2 \cdot (1-p)^{10} \\ p_{\boldsymbol{u}=110110/\boldsymbol{v}'} &= p^3 \cdot (1-p)^9 \\ p_{\boldsymbol{u}=110101/\boldsymbol{v}'} &= p^4 \cdot (1-p)^8 \\ p_{\boldsymbol{u}=110100/\boldsymbol{v}'} &= p^4 \cdot (1-p)^8 \end{aligned}$$

f. **Non-survivors' prob**

$$1 - p^2 \cdot (1-p)^{10} - p^3 \cdot (1-p)^9 - 2p^4 \cdot (1-p)^8$$

g. $LLR_3 = ln(\frac{7}{4})$

$$\begin{aligned} u_3 = \ & 0 \ : \ 1 \text{ branch} \quad 1 \cdot \left\{p^2 \cdot (1-p)^{10}\right\} = .0035 \\ u_3 = \ & 1 \ : \ 3 \text{ branches} \quad 2 \cdot \left\{p^4 \cdot (1-p)^8\right\} + \left\{p^3 \cdot (1-p)^9\right\} = .0020 \end{aligned}$$

h. **Less confidence than APP, better than SOVA**

Soft Information from Constraints

Section 7.4

Block Codes aggregate many tiny codes

Each row in an arbitrary linear encoder's parity matrix can be viewed as a simple linear parity code.

- Constraints
  - $p_{x/\boldsymbol{y}}$ is the probability of x given that the *decision* based on $\boldsymbol{y}$ meets code/modulation constraints.
- $p_{x/constraints} \propto p_{x,constraints} = \underbrace{p_{intrinsic}}_{current\ x's} \cdot \underbrace{p_{extrinsic}}_{other\ x's}$
- For instance, the parity-check equation $\boldsymbol{v} \cdot H = \mathbf{0}$ provides $n - k$ **parity constraints.**
- So $p_{x/constraints}$ essentially means the MAP finds the x most likely to satisfy all these parity constraints.
- There are other types of constraints also:
  - **Equality constraints** – These basically recognize that any x (often a bit) must have common decision in every constraint in which it participates.
  - **Modulator constraints** – Only certain constellation-to-bit mappings may occur for particular modulator (BICM).
  - **Channel-Model constraints** – Certain bit/x combinations may not be likely, given a certain channel filter.
    - These sometimes have the name "turbo equalization."

Basics PRIOR to the constraint

- BSC for $i = 1,2,9$
  - **Before constraint**:

$$p(v_i, y_i) = p(y_i/v_i) \cdot p(v_i\ )$$

$$p(v_i, y_i) = \begin{cases} (1-p) \cdot (p_i) & y_i = 1, v_i = 1 \\ p \cdot (1-p_i) & y_i = 1, v_i = 0 \\ p \cdot p_i & y_i = 0, v_i = 1 \\ (1-p) \cdot (1-p_i) & y_i = 0, v_i = 0 \end{cases}$$

intrinsic Extrinsic

Info on i from $j \neq i$

- AWGN for $i = 1,2,9$
  - **Before constraint**:

$$p(v_i, y) = \begin{cases} \boxed{\underbrace{\frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2\sigma^2}(y-\sqrt{\bar{\mathcal{E}}_{\boldsymbol{x}}})^2}}_{p_{int}}} \cdot \boxed{\underbrace{p_i}_{p_{ext}}} & v_i = 1 \\ \boxed{\underbrace{\frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2\sigma^2}(y+\sqrt{\bar{\mathcal{E}}_{\boldsymbol{x}}})^2}}_{p_{int}}} \cdot \boxed{\underbrace{(1-p_i)}_{p_{ext}}} & v_i = 0 \end{cases}$$

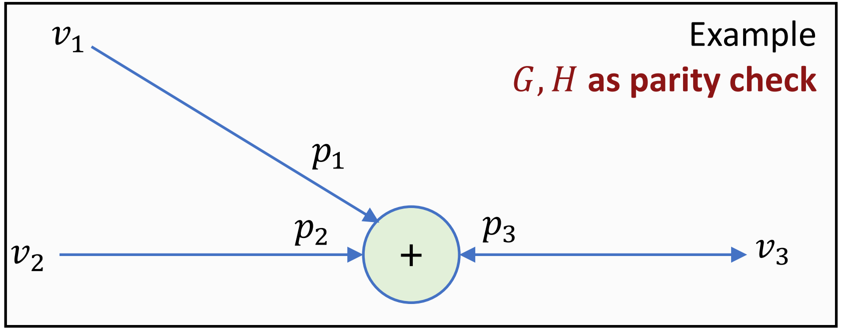
Example parity constraint

- Example has 3 bits in a specific **parity equation** (row of H, call it h_i, or column of H^t) ; $H \rightarrow G$.
  - **Generator $G's$ output** is such that $v_1 \oplus v_2 \oplus v_9 = 0$; this corresponds to 1's in positions 1,2, and 9 in a row of H.
  - First: A BSC with bit-error parameter p has channel outputs y_1 , y_2 , y_9 and encoder outs v_1 , v_2 , v_9 .

- S_E is a subset $S_E = \{\boldsymbol{v} \mid E(\boldsymbol{v}) = 0\}$ - all the bit combinations that satisfy the constraint:
  - $S_E = \{(0,0,0), (1,1,0), (1,0,1), (0,1,1)\}$

- $S_{E\backslash i}(y_i)$ fixes each set-codeword's position i to be the specific value y_i.
  - $S_{E\backslash 3}(y_3 = \mathbf{0}) = \{(0,0,0), (0,1,1)\}$

- MAP decoder to $v_{i=3}$ for this event satisfies $\max_{v_3 \in \{0,1\}} p_{v_{3/E}}$



$$p_{v_3\,/E} \propto p_{v_3,E} = p_{ext}(v_3\,/E, y_3) \cdot p_{int}(E, y_3)$$

$$p_{ext}(v_3\,/E, y_3) \propto \begin{cases} \Pr\{v_3 = y_3 = 0\} = p_1 \cdot p_2 + (1-p_1)\cdot(1-p_2) \\ \Pr\{v_3 = y_3 = 1\} = p_1 \cdot (1-p_2) + (1-p_1)\cdot p_2 \end{cases}$$

$$p_{int}(E, y_3) \propto \begin{cases} p_3 & ; y_3 = 1 \\ 1 - p_3 & ; y_3 = 0 \end{cases} \qquad p_3 = p_{BSC} = p$$

MAP Decoder maximizes $p_{v_i /E}$

- For bit $i = 3$:

$$p_{v_3,E} = \frac{1}{c_3'} \cdot \begin{cases} p_1 \cdot p_2 \cdot (1 - p_3) + (1 - p_1) \cdot (1 - p_2) \cdot (1 - p_3) & v_3 = 0 \\ p_1 \cdot (1 - p_2) \cdot\ p_3 + (1 - p_1) \cdot\ p_2 \cdot\ p_3 & v_3 = 1 \end{cases}$$

- For bit $i = 2$:

$$p_{v_2,E} = \frac{1}{c_2'} \cdot \begin{cases} p_1 \cdot p_3 \cdot (1 - p_3) + (1 - p_1) \cdot (1 - p_3) \cdot (1 - p_2) & v_2 = 0 \\ p_1 \cdot (1 - p_3) \cdot\ p_2 + (1 - p_1) \cdot\ p_2 \cdot\ p_3 & v_2 = 1 \end{cases}$$

- For bit $i = 1$:

$$p_{v_1,E} \frac{1}{c_1'} \cdot \begin{cases} p_3 \cdot p_2 \cdot (1 - p_1) + (1 - p_3) \cdot (1 - p_2) \cdot (1 - p_1) & v_1 = 0 \\ p_3 \cdot (1 - p_2) \cdot\ p_1 + (1 - p_3) \cdot\ p_2 \cdot\ p_1 & v_1 = 1 \end{cases}$$

Events and their probability calculation

- Satisfaction of parity check is an example of, more generally, **an event** $E(\boldsymbol{v}) = 0$.
- S_E is a subset $S_E = \{\boldsymbol{v} \mid E(\boldsymbol{v}) = 0\}$ - all the bit combinations that satisfy the constraint.
  - $S_{E\backslash i}(y_i)$ fixes each set-codeword's position i to be the specific value y_i.
- MAP decoder for this event satisfies $\max_{v_i \in \{0,1\}} p_{v_{i/E}}$.
- BSC/AWGN $\quad p_{ext}(y_i \,/\, E, y_i) = c_i \cdot \sum_{\boldsymbol{v} \in S_{E\backslash i}(y_i)} \prod_{\substack{j=1 \\ j \neq i}}^{n} p_j(E, y_i)$

$$\boxed{c_i = \left\{ \sum_{\boldsymbol{v} \in S_E} \prod_{j=1}^{n} p_j(E, y_i) \right\}^{-1}}$$

Recall L10:7

$$p_{ext}(v_3 \,/\, E, y_3) \propto \begin{cases} \Pr\{v_3 = y_3 = 0\} = p_1 \cdot p_2 + (1 - p_1) \cdot (1 - p_2) \\ \Pr\{v_3 = y_3 = 1\} = p_1 \cdot (1 - p_2) + (1 - p_1) \cdot p_2 \end{cases}$$

$$c_3 = \frac{1}{1 - 2 \cdot p_1 \cdot p_2}$$

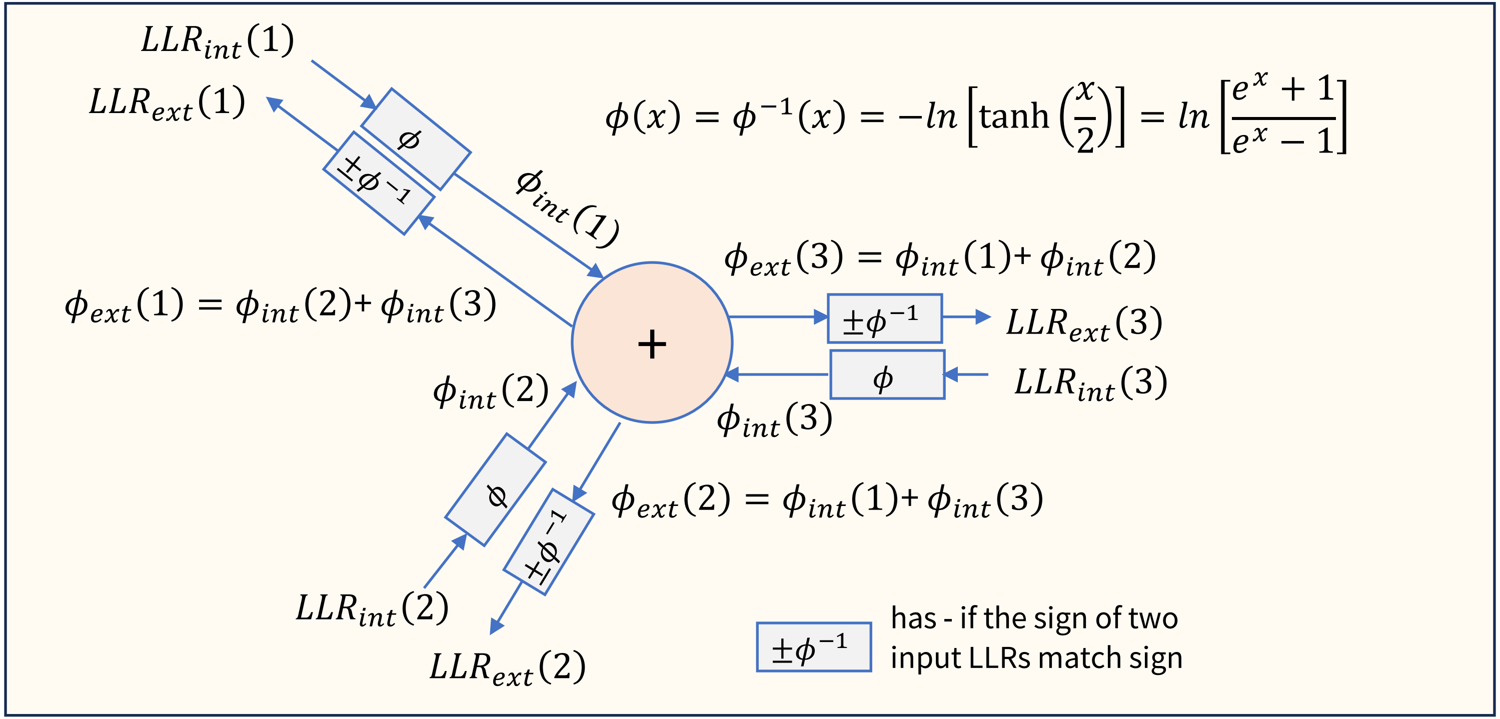
Similarly, for v_1 and v_2 - send p_{ext} to 3 other constraint decoders

Soft Bits

- The **soft bit** is $\chi_i = 2 \cdot Pr\{v_i = 0\} - 1 = 1 - 2 \cdot Pr\{v_i = 1\}$.
  - A soft bit accepts any probability (extrinsic , intrinsic, …) for $Pr\{v_i = 0\}$.
- The soft bit relates to LLR as $LLR_i = \ln \frac{\chi_i + 1}{\chi_i - 1}$ or $\chi_i = -tanh\left(\frac{LLR_i}{2}\right)$.
- By induction (with t_r = # of 1's in a row) $$\chi_i = \prod_{\substack{j=1 \\ j \neq i}}^{t_r} \chi_j \,.$$
- Use this soft bit with extrinsic information for all the "other" bits:
- Define the involution $$\boxed{\phi(x) = \phi^{-1}(x) = -\ln\left[\tanh\left(\frac{x}{2}\right)\right] = \ln\left(\frac{e^x + 1}{e^x - 1}\right)}$$
- So then $$\phi(LLR_{ext,i}) \triangleq +\ln\left(\frac{e^{LLR_{ext,i}} + 1}{e^{LLR_{ext,i}} - 1}\right) = -\ln\left(\tanh\left[\frac{LLR_{ext,i}}{2}\right]\right)$$
- And finally: $\chi_i \cdot \chi_j \leftrightarrow \phi(LLR_i) + \phi(LLR_j)$.
  - This means no multiplication, just adds and table look-up $\phi(x)$.

Illustration on next slide

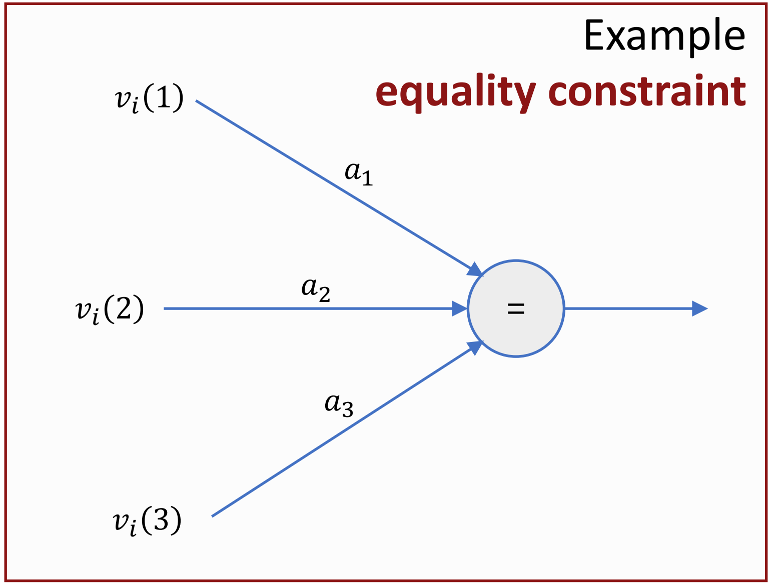
Parity Constraint Soft-Information Flows



- So each bit, considered like a tiny code, sends receives extrinsic info and sends intrinsic info, to all others in E.

Equality Constraints

- Each bit may participate in many constraints – it should ultimately have same value in them all.



$$S_E = \{(0,0,0),(1,1,1)\}$$

$$p_{ext}(v_2 \,/\, E, y_2) = c_2 \cdot \begin{cases} a_1 \cdot a_3 & v_i(2) = 1 \\ (1-a_1)\cdot(1-a_3) & v_i(2) = 0 \end{cases}$$

$$c_2 = \frac{1}{a_1 \cdot a_3 + (1-a_1)\cdot(1-a_3)}$$

$$p_{int}(E, y_2) \propto \begin{cases} p_2 & ;\, y_2 = 1 \\ 1-p_2 & ;\, y_2 = 0 \end{cases}$$

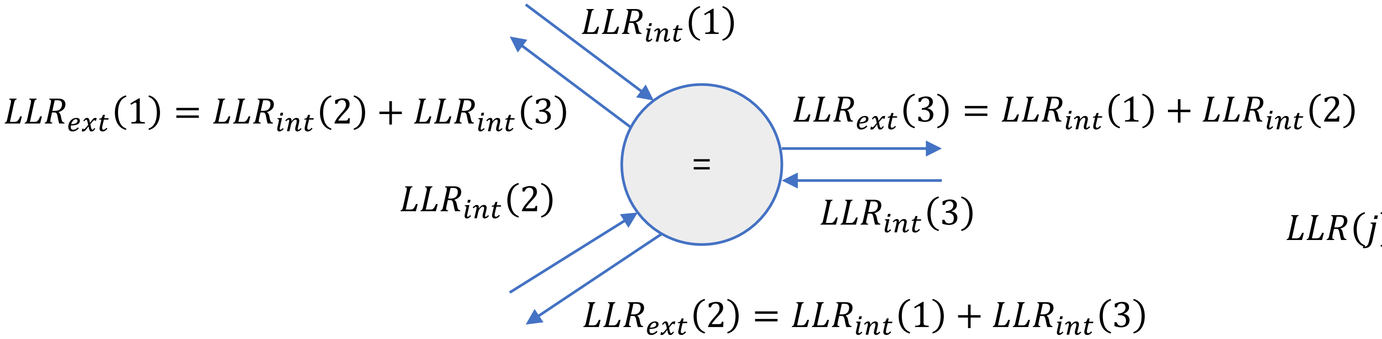
Equality-Constraint Decoder maximizes

$$p_{v_i,E} = c_i' \cdot \begin{cases} a_1 \cdot a_2 \cdot a_3 & v_i = 1 \\ (1-a_1)\cdot(1-a_2)\cdot(1-a_3) & v_i = 0 \end{cases}$$

$$c_i' = \frac{1}{a_1 \cdot a_2 \cdot a_3 + (1-a_1)\cdot(1-a_2)\cdot(1-a_3)}$$

Equality Constraint Soft-Information Flows

- The extrinsic information returns to other (e.g., parity) constraints, and the constraint accepts intrinsic from others

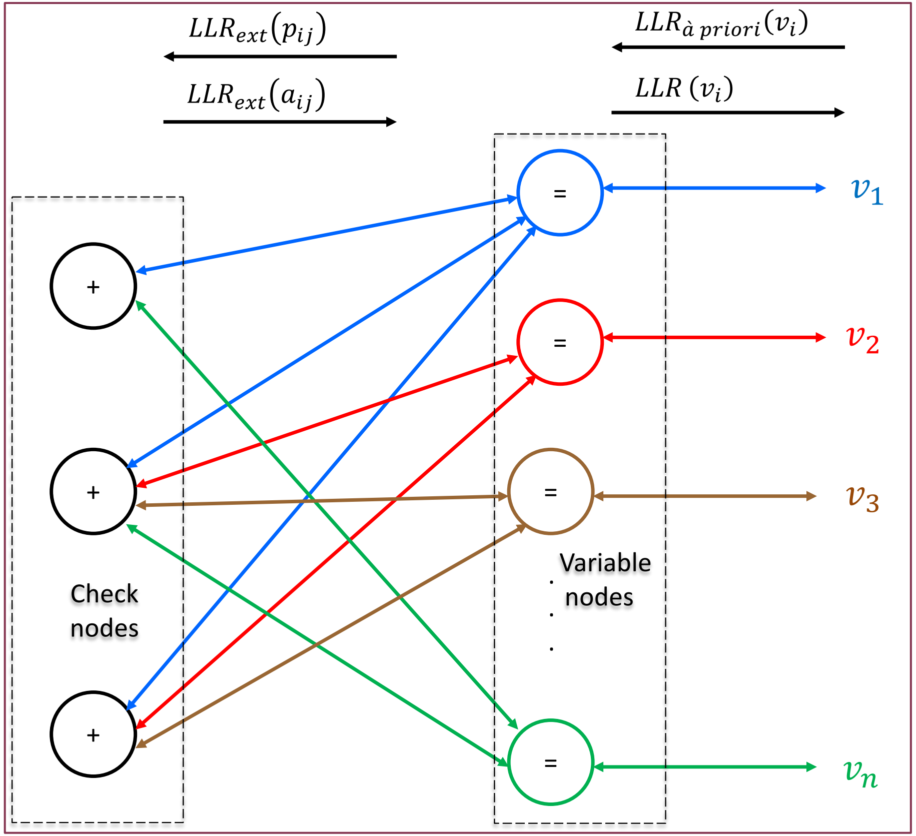


$$LLR(j) = \sum_{\substack{j=1 \\ j \neq i}}^{t_c} LLR(i)$$

- The Equality and Parity constraints for a binary block code can thus cycle soft information.
- This is another form of iterative decoding.

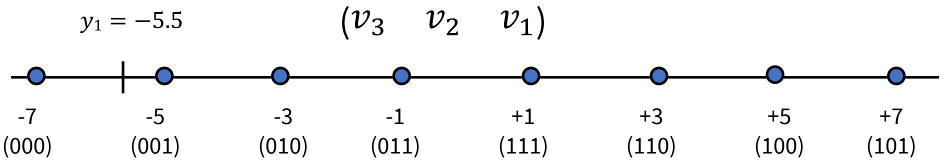
Simple Iterative Decoder Illustration

- It's called a "Tanner Graph" or "Factor Graph."
- Decoding may take multiple iterations:
  - When extrinsic data from an equality node cycles back to that same node, the soft-information can become "biased."
  - Such a biased decoder then loses exact MAP quality.
- Good codes try to make the cycle longer than the number of iterations that lead to convergence.
  - This can only be done approximately in practice.
- Good LDPC codes achieve this.
  - Designers actually design the H matrix .
  - And then just use a corresponding systematic G.
    - Do this by simple row add operations to designed H to H_{sys} =$[h \quad I]$.
    - G= $[I \quad h^t]$ so then $G \cdot H^t = G \cdot H_{sys}^t = 0$.



Soft-Information from constellation

- Example for 1 dimension of Gray Code:
  - E.g., 64QAM, y_1



- Basically, sum contributions for common v_i values of 0 and then 1, normalizing the constant as follows:

$$
\begin{aligned}
p(y_1=-5.5, v_3=0) &= c_1 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(.5)^2} + e^{-\frac{1}{2\sigma^2}(1.5)^2} + e^{-\frac{1}{2\sigma^2}(2.5)^2} + e^{-\frac{1}{2\sigma^2}(4.5)^2} \right) \cdot (1-p_3) \\
p(y_1=-5.5, v_3=1) &= c_1 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(6.5)^2} + e^{-\frac{1}{2\sigma^2}(8.5)^2} + e^{-\frac{1}{2\sigma^2}(10.5)^2} + e^{-\frac{1}{2\sigma^2}(12.5)^2} \right) \cdot p_3 \quad , \\
p(y_1=-5.5, v_2=0) &= c_2 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(.5)^2} + e^{-\frac{1}{2\sigma^2}(1.5)^2} + e^{-\frac{1}{2\sigma^2}(10.5)^2} + e^{-\frac{1}{2\sigma^2}(12.5)^2} \right) \cdot (1-p_2) \\
p(y_1=-5.5, v_2=1) &= c_2 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(2.5)^2} + e^{-\frac{1}{2\sigma^2}(4.5)^2} + e^{-\frac{1}{2\sigma^2}(6.5)^2} + e^{-\frac{1}{2\sigma^2}(8.5)^2} \right) \cdot p_2 \quad , \\
p(y_1=-5.5, v_1=0) &= c_3 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(1.5)^2} + e^{-\frac{1}{2\sigma^2}(2.5)^2} + e^{-\frac{1}{2\sigma^2}(8.5)^2} + e^{-\frac{1}{2\sigma^2}(10.5)^2} \right) \cdot (1-p_1) \\
p(y_1=-5.5, v_1=1) &= c_3 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(.5)^2} + e^{-\frac{1}{2\sigma^2}(4.5)^2} + e^{-\frac{1}{2\sigma^2}(6.5)^2} + e^{-\frac{1}{2\sigma^2}(12.5)^2} \right) \cdot p_1 \quad .
\end{aligned}
$$

LDPC Codes

Section 8.3.3

LDPC as "almost random" codes

- R. Gallager (MIT), early 1960's, designed the parity check matrix H directly (e.g., design the null space / checks).
  - His code ensemble averaged $n - k$ parity bits that were randomly placed in H for given large n.
    - $r = k/n$ is finite.
    - The "low density" (LD) part ➔ **SPARSE BINARY MATRIX** (see matlab's "sparse.m" and "nnz.m" commands).
- The ensemble works at capacity limit; RG even found some codes that were really good.
- However, the consequent ML Decoders however were too complex!
- 1990's – Turbo/Iterative Decoding suggests revisit of LDPC codes.
  - The decoders were feasible to implement 30 years later, reviving LDPC.
- 2020's – LDPC codes find heavy use in modern designs.
  - 5G Wireless
  - Wi-Fi 5, 6, 7
  - High-speed Fiber
- Polar Codes (Arikan) – 2009 (use another suboptimal "successive-decoding" method).
  - Even better for binary AWGN, but the BICM-ID does not work with PC's successive decoding and limits polar codes' applicability.

Some H-Related Definitions

- **4 Cycle** – two rows have at least two 1's in same columns.
  - This is not good. Why?
  - Equality constraint → to parity → equality → parity → back again!
  - Biases accumulate quickly in constraint-based iterative decoding.

$$\begin{array}{cc} (n-k) & \begin{bmatrix} 1 & 0 & \cdots & \cdots & 1 & 0 \\ 0 & \color{darkred}{\mathbf{1}} & 0 & \color{darkred}{\mathbf{1}} & 0 & 0 \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 1 & 0 \\ 0 & \color{darkred}{\mathbf{1}} & \cdots & \color{darkred}{\mathbf{1}} & \ddots & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \\ & n \end{array}$$

- **Regular Parity Matrix** (sparse)
  - All rows have t_r 1's.
  - All columns have t_c 1's.
  - So $(n-k) \cdot t_r = k \cdot t_c$.
  - Otherwise, it is an **irregular** parity matrix.

$$\boxed{r = 1 - \frac{t_c}{t_r}}$$

- **Density-Limit bound:**
  - avoids all 4 cycles,
  - ensures sparse H for finite r, &
  - basically means n will be large.

$$n \leq \underbrace{\binom{(n-k)}{2}}_{\text{choose 2 rows}} \Big/ \underbrace{\binom{t_c}{2}}_{\text{choose 2 cols}} = \frac{(n-k)\cdot(n-k-1)}{t_c \cdot (t_c - 1)}$$

Large n helps for "random coding" also ; so, how can a designer get such a code with implementable decoder? (typical $n > 1000$)

Some LDPC Design Choices

Name	Quasi-Cyclic	Generic Irregular	Application Specific
Reg/irreg	regular	Slightly irregular	irregular
Uses	Wi-Fi	General	5G, DVB
positives	Matlab functions	379A class Matlab, no restrictions Good for M'ary	Puncturing Parallelism Special matlab
negatives	Not quite optimum	Not as well known/supported	Perhaps too specific

- There can be an SNR (equivalently r) dependence.
- Designers don't really want to design a new code for each channel.
- The code's amenability to puncturing/rate-variation is important.

Shaping Gain Offset

- **Review Lecture 6**
  - Turbo, LDPC, polar, …
  - DO NOT ADDRESS **Shaping Gain**

$$\gamma \triangleq \frac{\left(d^2_{\min}(\boldsymbol{x})/\bar{\mathcal{E}}_{\boldsymbol{x}}\right)}{\left(d^2_{\min}(\breve{\boldsymbol{x}})/\bar{\mathcal{E}}_{\breve{\boldsymbol{x}}}\right)} = \underbrace{\frac{\left(\frac{d^2_{\min}(\boldsymbol{x})}{V^{2/N}(\Lambda)}\right)}{\left(\frac{d^2_{\min}(\breve{\boldsymbol{x}})}{V^{2/N}(\breve{\Lambda})}\right)}}_{\substack{\gamma_f \\ \text{fundamental gain}}} \cdot \underbrace{\frac{\left(\frac{V^{2/N}(\Lambda)}{\bar{\mathcal{E}}_{\boldsymbol{x}}}\right)}{\left(\frac{V^{2/N}(\breve{\Lambda})}{\bar{\mathcal{E}}_{\breve{\boldsymbol{x}}}}\right)}}_{\substack{\gamma_s \\ \text{shaping gain}}} \left\{ 0 < \gamma_s < 1.53 \right.$$

This is what the LDPC/Turbo works improve.
Eb/N0 only equals SNR when $r = 1/2$.

- See Section 8.5 for shaping codes
  - Can get up to 1.2 dB of the 1.53 dB
  - $\gamma_{s,offset}$ is shaping gain for particular constellation size (or $\bar{b}$)

(t_c, t_r)	$\bar{b}$	γ_s offset	deviation from $\mathcal{C}_{\lvert C\rvert=2}$
(3,6)	.5	.184 dB	1.1 dB
(4,8)	.5	.184 dB	1.6 dB
(5,10)	.5	.184 dB	2.0 dB
(3,5)	.4	.051 dB	1.3 dB
(4,6)	1/3	.033 dB	1.4 dB

Regular codes' cannot get to capacity:
Richardson/Urbanke, $\gamma_{s,offset}$ added here

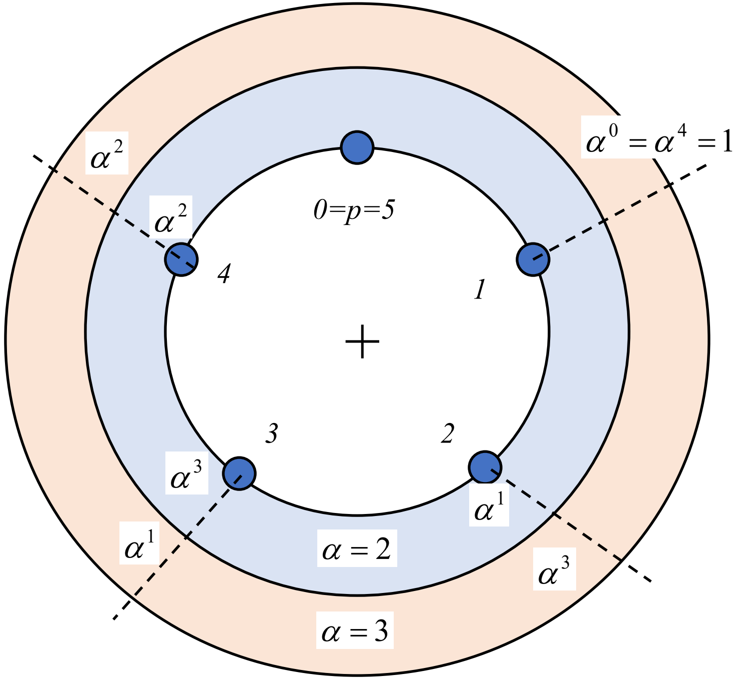
$$\gamma_{s,offset} = \begin{cases} 0.1 \cdot \bar{b} \text{ dB} & 0 \le \bar{b} \le 0.33 \\ 0.27 \cdot \bar{b} - .057 \text{ dB} & 0.33 \le \bar{b} \le 0.4 \\ 1.33 \cdot \bar{b} - .48 \text{ dB} & 0.4 \le \bar{b} \le 0.5 \\ 0.2 \cdot \bar{b} + .084 \text{ dB} & 0.5 \le \bar{b} \le 1 \\ 1 \cdot \bar{b} - .72 \text{ dB} & 1 \le \bar{b} \le 2 \\ 0.2 \cdot \bar{b} + .85 \text{ dB} & 2 \le \bar{b} \le 3 \\ 0.17 \cdot \bar{b} + .83 \text{ dB} & 3 \le \bar{b} \le 4 \\ 1.53 \text{ dB} & \bar{b} \ge 4 \end{cases}$$

Galois Field p reminder from L6:24-25

- $GF(p) = \{0,1,\ldots,p-1\}$
  - Addition is modulo p.
  - Multiplication is close, with division defined by inverse, and follows from any prime element $\alpha \in GF(p)$.

$$GF(5) = \{0 \;\; 1 \;\; \alpha \;\; \alpha^2 \;\; \alpha^3\}$$

GF exists for any prime p or product of such primes.



$\times$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

α	α^2	α^3	α^4
2	4	3	1
3	4	2	1

Prelude to Quasi-Cyclic LDPC

- LDPC Design Goals:
  - Design avoids 4-cycles.
  - H should have rank $n-k$.
  - H should have low density of 1's.

- Design should have good performance (including all neighbors at all distances),
  - but still have some structure to help encoder and especially decoder implementation.

- W is a **Latin-Square Matrix**.
  - Each row/col contains each set element once.
  - W is clearly nonsingular.

- J is a **right-shift matrix** (applied to row vector) that
  - circularly shifts rows of matrix (on right) to the right, $p \times p$.

- H is the **dispersion** of W using J over $GF(p)$ that
  - replaces each element $w_{i,j}$ by $(p-1)\times(p-1)$ matrix $J^{w_{i,j}}$.

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

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

$$H = \begin{bmatrix} I & J & J^2 \\ J & J^2 & I \\ J^2 & I & J \end{bmatrix}$$

Quasi-Cyclic LDPC

- Special Latin-Square W $(p \times p)$ matrix for any $\eta \in GF(p)$:

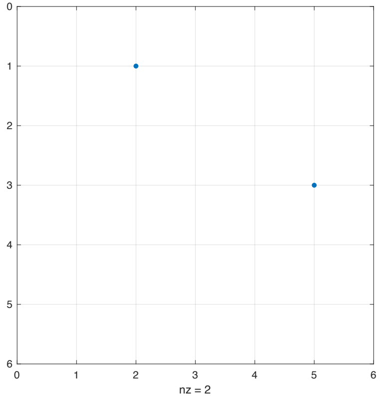
$$W = \begin{bmatrix} \eta - \alpha^0 & \eta - \alpha & \cdots & \eta - \alpha^{p-2} & \eta \\ \alpha \cdot \eta - \alpha^0 & \alpha \cdot \eta - \alpha & \cdots & \alpha \cdot \eta - \alpha^{p-2} & \alpha \cdot \eta \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha^{p-2} \cdot \eta - \alpha^0 & \alpha^{p-2} \cdot \eta - \alpha & \cdots & \alpha^{p-2} \cdot \eta - \alpha^{p-2} & \alpha^{p-2} \cdot \eta \\ -\alpha^0 & -\alpha^1 & \cdots & -\alpha^{p-2} & 0 \end{bmatrix}$$

- W's dispersion (with $(p-1) \times (p-1)$ J) is the QC-LDPC matrix H and has no 4-cycles (Zhang et al.).
  - Usually $p = 2^m$.
- These codes are regular (because of the J matrix and its shifts).
- Matlab ldpcQuasiCyclicMatrix.m command produces these:
  - Inputs are $p-1$ and W (which is a Latin-Square matrix with rules on how to create it).

QC-LDPC codes and Sparse matrices

```
>> i=[1 3]; % rows
>> j=[2 5];  % columns
>> v=[1 1]; % values to insert
>> S=sparse(i,j,v,5,5)
  (1,2)       1
  (3,5)       1
>> nnz(S) =    2
>> full(S)  =
     0    1    0    0    0
     0    0    0    0    0
     0    0    0    0    1
     0    0    0    0    0
     0    0    0    0    0
```

>> spy(S)



- ldpc..ze is for large spare matrices, like LDPC matrices H.

```
% Wi-FI code 802.11 (Wi-Fi 5,6,7 )'s parity-check matrix with r=3/4 LDPC
   P = [
      16 17 22 24  9  3 14 -1  4  2  7 -1 26 -1  2 -1 21 -1  1  0 -1 -1 -1 -1
      25 12 12  3  3 26  6 21 -1 15 22 -1 15 -1  4 -1 -1 16 -1  0  0 -1 -1 -1
      25 18 26 16 22 23  9 -1  0 -1  4 -1  4 -1  8 23 11 -1 -1 -1  0  0 -1 -1
       9  7  0  1 17 -1 -1  7  3 -1  3 23 -1 16 -1 -1 21 -1  0 -1 -1  0  0 -1
      24  5 26  7  1 -1 -1 15 24 15 -1  8 -1 13 -1 13 -1 11 -1 -1 -1 -1  0  0
       2  2 19 14 24  1 15 19 -1 21 -1  2 -1 24 -1  3 -1  2  1 -1 -1 -1 -1  0
     ];  % 6 x 24 matrix
   blockSize = 27;
>>  H = ldpcQuasiCyclicMatrix(blockSize, P); % creates dispersion of P with J_27
>> size(H) =   162   648
```

```
% Example 1:
   blockSize1 = 3;
   P1 = [0 -1 1 2; 2 1 -1 0];
   H1 = ldpcQuasiCyclicMatrix(blockSize1,P1)
  6 x12 sparse logical array

   (1,1)     1
   (5,1)     1
   (2,2)     1
   (6,2)     1
   (3,3)     1
   (4,3)     1
   (6,4)     1
   (4,5)     1
   (5,6)     1
   (3,7)     1
   (1,8)     1
   (2,9)     1
   (2,10)    1
   (4,10)    1
   (3,11)    1
   (5,11)    1
   (1,12)    1
   (6,12)    1
>> size(H1) =    6    12
>> full(H1) =
   1  0  0 | 0  0  0 | 0  1  0 | 0  0  1
   0  1  0 | 0  0  0 | 0  0  1 | 1  0  0
   0  0  1 | 0  0  0 | 1  0  0 | 0  1  0
  ---------------------------------------
   0  0  1 | 0  1  0 | 0  0  0 | 1  0  0
   1  0  0 | 0  0  1 | 0  0  0 | 0  1  0
   0  1  0 | 1  0  0 | 0  0  0 | 0  0  1
```

QC-LDPC encoder and decoder

- With the H matrix, create objects with ldpcEncoderConfig and ldpcDecoderConfig
  - Encode
  - Decode

```
>> wificonf=ldpcEncoderConfig(H)
    ParityCheckMatrix: [162 × 648 logical]
      Read-only properties:
        BlockLength: 648
      NumInformationBits: 486
      NumParityCheckBits: 162
          CodeRate: 0.7500
 wificonfdec=ldpcDecoderConfig(H,'norm-min-sum')
 ldpcDecoderConfig with properties:
    ParityCheckMatrix: [162 × 648 logical]
         Algorithm: 'norm-min-sum'
   Read-only properties:
        BlockLength: 648
    NumInformationBits: 486
    NumParityCheckBits: 162
          CodeRate: 0.7500
      NumRowsPerLayer: 27>> Y=ldpcEncode(X,wificonf);

>> X=prbs(7,486)';
>> Y=ldpcEncode(X,wificonf);
>> X1=ldpcDecode(1-2*Y,wificonfdec,6);
>> biterr(X,X1) =    0
```

```
>> error = [ 1 zeros(1,49) 1 zeros(1,49) 1 zeros(1,99) 1 zeros(1,45) 1 zeros(1,61)];
>> errorldpc=[error, error, zeros(1,32)];
>> X1=ldpcDecode(1-2*(Y+errorldpc'),wificonfdec,6);
>> biterr(X,X1) =    0
```

☠ Warning: I could not get the ‘bp’ (Belief Propagation) option for ldpcDecode to work with noise UNLESS the errorldpc/noise scales by <0.9; I think this relates to soft-info scaling internal to “bp” option

2nd decoder input can be 'bp', 'layered-bp’, 'norm-min-sum', or 'offset-min-sum' and the corresponding algorithms are belief propagation decoding, layered belief propagation decoding, normalized min-sum decoding, and offset min-sum decoding respectively.
https://www.mathworks.com/help/comm/ref/ldpcdecode.html

- You can begin to experiment now:
  - The decoder input is “LLR,” so you could:
  - compute from a Gray mapped constellation,
  - run for different SNR,
  - compute error curves,
  - etc

Generic Irregular Codes

- Thanks go to E. Eleftheriou and S. Olcer of IBM (> 20 years so public domain 😀).
  - These use the shift-matrix dispersion concept and in easier way with $p \times p$ shift matrix J.
  - Their design checks for 4-cycles and linear-dependence → irregular codes.
    - Their construction deletes any row that causes 4 cycle or linear dependence on previous rows.
  - The call the number of deleted rows m when the desired $n-k$ linearly independent rows is achieved.

- Starts with desired t_r and t_c
  - Eventually $n-k < t_c - p$

$$\tilde{t}_c \triangleq (t_c - 1)\cdot m + t_c \cdot \left(\frac{n-m}{n}\right) \qquad r = 1 - \frac{\tilde{t}_c}{t_r}$$

$$H = \begin{bmatrix} I & I & \cdots & I & I \\ I & J & J^2 & \cdots & J^{t_r-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ I & J^{\tilde{t}_c-1} & J^{2(\tilde{t}_c-1)} & \cdots & J^{(t_r-1)(\tilde{t}_c-1)} \end{bmatrix}$$

(n,k)	m	p	t_c	t_r	r	$\gamma_{s,offset}$ (8.71)	Γ at 10^{-7}	$\gamma_{f,eff}$ at 10^{-7}
(276,209)	2	23	3	12	.7572	0.69 dB	4.5 dB	5.1 dB
(529,462)	2	23	3	23	.8733	0.93 dB	3.9 dB	5.7 dB
(1369,1260)	2	37	3	37	.9204	1.01 dB	3.3 dB	6.3 dB
(2209,2024)	3	47	4	47	.9163	1.01 dB	2.8 dB	6.8 dB
(4489,4158)	4	67	5	67	.9263	1.03 dB	2.5 dB	7.1 dB
(7921,7392)	5	89	6	89	.9332	1.04 dB	2.3 dB	7.3 dB

Table 8.21: Generic LDPC code parameters.

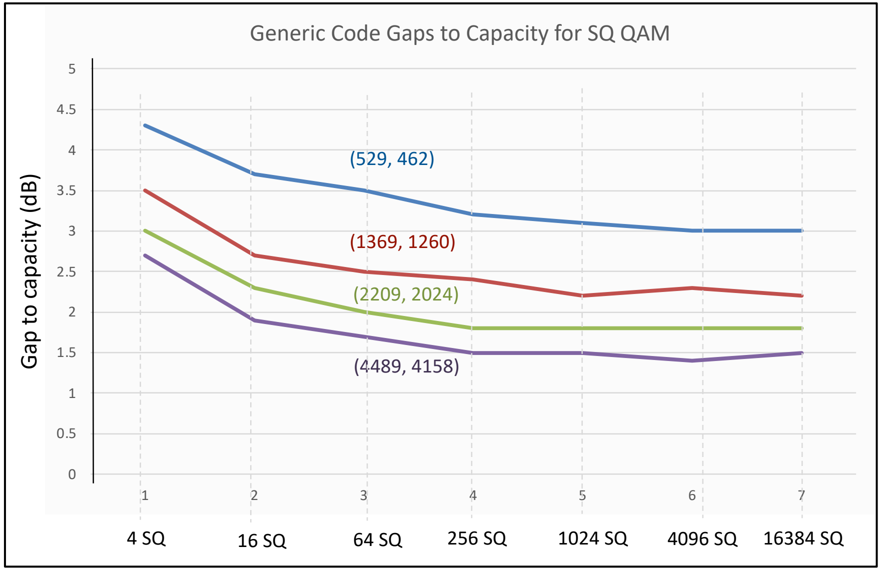
Generic Software (customized to 379A)

- To get H (not yet in sparse format)

```
function [H_no_dep H] = get_h_matrix(p,tr,tc,first_1);
 Generate LDPC H Matrix Uses Generic-LDPC Method As Per Cioffi's Class Notes
 Example: to Generate (529,462) code, p=23, rw=23, cw=3, first_1=2
      H = get_h_matrix(23,23,3,2);
----------------------------------------
 Definition of input variables
 p     :  Prime number of the size of base matrix of size p-by-p
 tr    :  Row weight = # of base matrices (or 1's) /row, equivalent to K
 tc    :  Col weight = # of base matrices (or 1's) per column,eq to J
 first_1:  Set to 2 in generic LDPC code, so right shift by first_1-1
-------------------------------------
 Definition of output variables
 H_no_dep  : the parity check matrix with no dependent rows
 H         : without removing the dependent rows
--------------------------------
EE379A, Chien-Hsin Lee, first version 06/2006, edits by J. Cioffi since
```

```
>> H = get_h_matrix(23,23,3,2);
>> size(H) =    67   529
>> 529-67  =   462
>> H=nonsinglastnk(H);
>> generic=ldpcEncoderConfig(logical(sparse(H)))
    ParityCheckMatrix: [67 × 529 logical]
BlockLength: 529
   NumInformationBits: 462
   NumParityCheckBits: 67
          CodeRate: 0.8733
```

```
>> X=prbs(7,462);
>> Y=ldpcEncode(X',generic);
>> genericdec=ldpcDecoderConfig(generic,"norm-min-sum");
>> errorgeneric=[error , 1 zeros(1, 99), 1 1 zeros(1,98) zeros(1,21)];
>> size(errorgeneric) % =     1   529
>> X1=ldpcDecode(1-2*(Y+1*errorgeneric'),genericdec,6);
>> biterr(X',X1) % =     0
```



Other Irregular

- Digital Video Broadcast standard has:
  - r =1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, or 9/10
  - $n = 64{,}800$
  - Can then use ldpcencode.m and ldpcdecode.m .

>> Hdvb=dvbs2ldpc(r)

- 5G standard (for 5G’s live data, not %G’s control channel):
  - has good puncturing, parallelism, and gain (see slides 27,28)
  - Is specific to this application, but may be good elsewhere also.
  - Matlab commands are

>> nrLDPCEncode.m
>> nrLDPCDecode.m

5G Code

- Using same "lifting" (Latin Squares) except with all-zeros matrices also allowed in some positions (so $\mathcal{J} \rightarrow \{\mathcal{J}, \mathbf{0}\} = Z$).
  - Many forms of the Z matrices to be lifted that use two "base matrices."
- Former 379 student Rick Wesel (now UCLA Prof) contributed concepts that allow:
  - Scalable decoder complexity with rate choice over wide range from 1/5 to 1/3
  - See reference [7] in Ericsson article below.
- See tutorial articles by
  1. Qualcomm: Tom Richardson and Shrinivas Kudekar, "Design of Low-Density Parity Check Codes for 5G New Radio." IEEE Communications Magazine (Volume: 56, Issue: 3, March 2018), **pp.** 28 - 34, **DOI:** https://ieeexplore.ieee.org/document/8316763 .
  2. Ericsson: Dennis Hui et al, "Channel Coding in 5G New Radio," IEEE Vehicular Technology Magazine (Volume: 13, Issue: 4, December 2018), 60 - 69, **DOI:** 10.1109/MVT.2018.2867640 .
- More parity bits sent upon CRC failure (see L11).
  - Complexity scales with N (rate increase)
  - Unlike puncturing with turbo codes

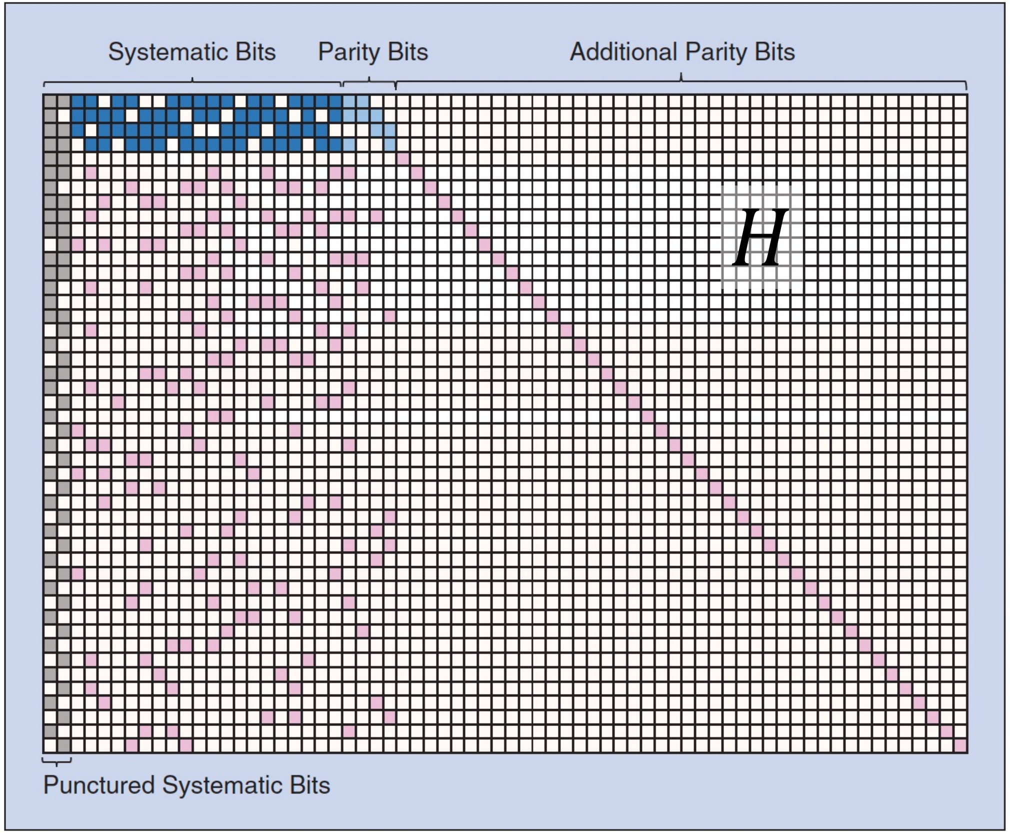


FIGURE 2 The structure of NR LDPC base matrix 1. Each square corresponds to one element in the base matrix or a $Z \times Z$ subblock in the PCM.

More 5G codes (Ericsson paper)

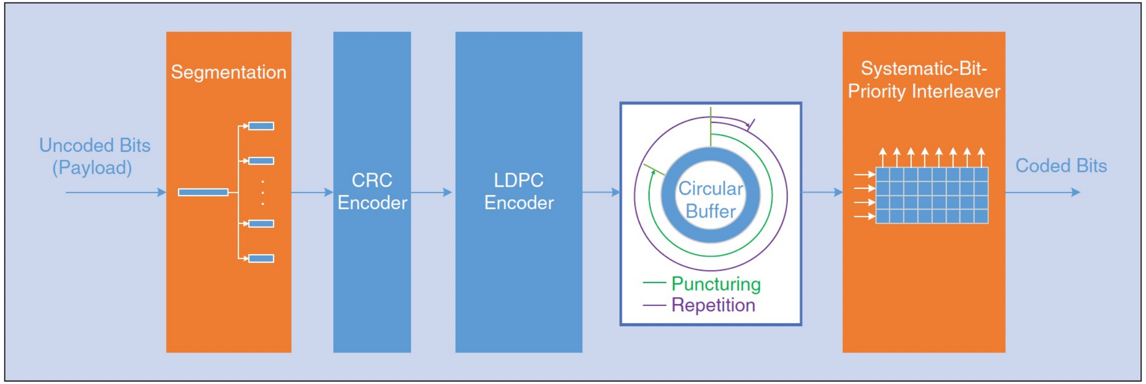


FIGURE 1 The NR LDPC coding chain.

TABLE 1 *NR LDPC base matrix parameters.*

Parameter	*Base Matrix 1*	*Base Matrix 2*
Minimum design code rate	1/3	1/5
Base matrix size	46 × 68	42 × 52
Number of systematic columns	22	10
Maximum information block size K	8,448 (= 22 × 384)	3,840 (= 10 × 384)
Number of nonzero elements	316	197

- 5G mandates base code use by rate and K

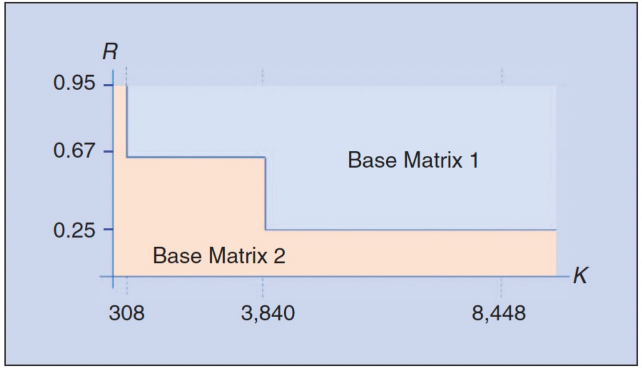


FIGURE 3 The usage of the two base matrices specified for the NR data channel. For K larger than the maximum information block size, code block segmentation is applied.

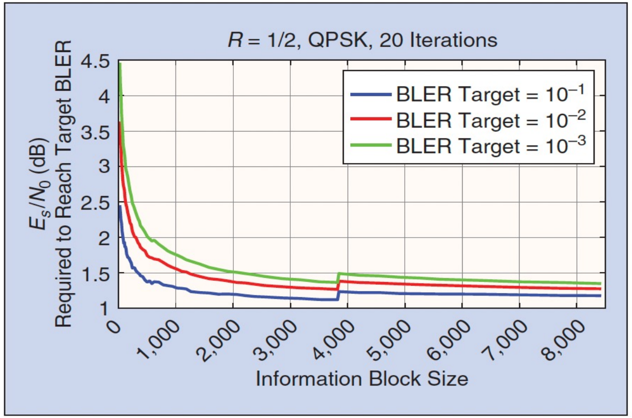


FIGURE 4 The performance of NR LDPC codes at code rate 1/2 for QPSK modulation.

Polar Codes

Section 8.3.4

Polar Codes Brief Commentary

- Polar Codes – Positives (Arikan); PC are:
  - not random,
  - based on essentially finite-field Fourier Transform size n ,
  - have simpler suboptimal decoders (successive decoders) ,
  - smaller gap for finite n,
  - used for binary (BPSK) control channel in 5G, &
  - lower delay.

- Polar Codes – Negatives:
  - Code design strong depends on SNR, instead of puncturing.
  - The successive decoder is not really compatible with M'ary QAM.
  - PC don't provide that much more gain.

Limited course time and likely studied in EE387 course

Not (yet) heavily used

So not in 379's

There are much bigger impacts to performance that arise from
a). Handling ISI/filtering – A and B
b). Optimizing transmit spectra, B
c). Allocation of dimensions/energy to multiple users sharing channel (B)

GRAND Decoders (L12) get same or better gain for simple block codes used as product codes, with yet lower decoder computation.

STANFORD

End Lecture 10

backup

$$p\,(v_i) = \begin{cases} Pr\{v_i = 0\} = \dfrac{(1-a_1)\cdot(1-a_2)\cdot(1-a_3)}{a_1\cdot a_2\cdot a_3 + (1-a_1)\cdot(1-a_2)\cdot(1-a_3)} \\ \Pr\{v_i = 1\} = \dfrac{a_1\cdot a_2\cdot a_3}{a_1\cdot a_2\cdot a_3 + (1-a_1)\cdot(1-a_2)\cdot(1-a_3)} \end{cases}$$

- sd