Lecture 12

Guessing Decoders and Product Codes

February 20, 2024

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

Instructor EE379A – Winter 2024

Announcements & Agenda

- Announcements
  - PS5 due tomorrow.

- Problem Set 6 = PS6 due Wednesday February 28
  1. 3.3 379 Model
  2. 3.4 Bias and SNR
  3. 3.6 Noise Enhancement
  4. 3.21 ISI Quantification
  5. 3.24 Peak Distortion

- Today
  - Retransmission – Error Detection
  - Guessing Random Additive Noise Decoding
  - GRAND Soft Information
  - Product Codes & GRAND
  - Matlab software (MIT education-use permitted)

Hot Off the Presses

Retransmission – Error-Detecting Codes

Section 8.6.3

CRC Error Detection and Retransmission

- Cyclic Redundancy Check codes are (usually) binary and only detect errors (so $s(D) \neq 0$).
  - CRCs mostly use simple binary versions of the previous encoders/decoders.
  - Table below lists some with $d_{free} = 4$ and $n_{max} = 2^{2^r} - 1$.
- These detect:
  - all single and 2-bit errors, and also any odd number of bit errors. The $(D + 1)$ factor forces even distance between codewords.
  - any burst of length $\leq n - k$ (because this is the length of $g(D)$ - such a burst is not divisible by $g(D)$).

Name	$g(D)$	factored
CRC-7	$D^7 + D^6 + D^4 + 1$	$(D^4 + D^3 + 1) \cdot (D^2 + D + 1) \cdot (D + 1)$
CRC-8	$D^8 + D^2 + D + 1$	$(D^7 + D^6 + D^5 + D^4 + D^3 + D^2 + 1) \cdot (D + 1)$
CRC-12	$D^{12} + D^{11} + D^3 + D^2 + D + 1$	$(D^{11} + D^2 + 1) \cdot (D + 1)$
CRC-16 USA	$D^{16} + D^{15} + D^2 + 1$	$(D^{15} + D + 1) \cdot (D + 1)$
CRC-16 Euro	$D^{16} + D^{15} + D^5 + 1$	$(D^{15} + D^{14} + D^{13} + D^{12} + D^4 + D^3 + D^2 + D + 1) \cdot (D + 1)$
CRC-24	$D^{24} + D^{23} + D^{14} + D^{12} + D^8 + 1$	$(D^{10} + D^8 + D^7 + D^6 + D^5 + D^4 + D^3 + D + 1) \cdot (D^{10} + D^9 + D^6 + D^4 + 1) \cdot (D + 1)$
CRC-32	$D^{32} + D^{26} + D^{23} + D^{22} + D^{16} + D^{12} + D^{11} + D^{10} + D^8 + D^7 + D^5 + D^4 + D^2 + D + 1$ (appears prime, not sure)	

Analysis – CRCs are for detection ONLY.

- $P_u \triangleq$ **undetected** error probability $P_u < 2^{k-n} \cdot (\bar{P}_b)^4$; $s = 0$ for wrong codeword.

Name	$P_u/(\bar{P}_b)^4$	$1 - P_u/(\bar{P}_b)^4$	Reliability
CRC-7	2^{-7}	.99221875	2 nines
CRC-8	2^{-8}	.99609375	3 nines
CRC-12	2^{-12}	.999755859375	4 nines
CRC-16 USA	2^{-16}	0.999984741210938	5 nines
CRC-16 Euro	2^{-16}	0.999984741210938	5 nines
CRC-24	2^{-24}	0.999999940395355	7 nines
CRC-32	2^{-32}	0.999999999767169	9 nines

High $\bar{P}_b$ if inner code fails

$\left.\begin{matrix}\\\\\end{matrix}\right\}$ *voice reliability*)

{*video reliability*

$\left.\begin{matrix}\\\\\end{matrix}\right\}$ *core network reliability*)

{*critical reliability*

{*storage*

- These are link-layer reliabilities - $\bar{P}_b$ could be high within a CRC codeword if large—N inner-code fails,
  - but still, even if $\bar{P}_b$=.1, these get very low.
- TCP-IP and higher-level session/application CRC checks (possibly using RS codes for detection) would create super reliability with "once in a century" level failures.
- These are cyclic codes so use earlier simple generators and receiver-syndrome calculation circuits.

Retransmission

- **Automatic Repeat Request (ARQ)**: If the CRC detects an error, resend the codeword.
- ARQ requires a mechanism for acknowledgment (back-channel) or ACK/NAK.
  - The NAK returns upon the receiver's non-zero CRC syndrome calculation.
  - P_c is the correct receipt probability (syndrome is zero).

$$\mathbb{E}[L_{retrans}] = \sum_{l=1}^{\infty} l \cdot P_c \cdot (1 - P_c)^l = \frac{1}{P_c}$$

- Throughput = $\left(\frac{k}{n}\right) \cdot P_c \cdot R$ bps

- Throughput represents the "real data rate" with code redundancy and retransmission accounted.
  - Throughput assumes infinite buffer delay is possible.

- There are entire courses in this network/queuing area, see EE384S (Bambos, Spring Q).

Hybrid ARQ (HARQ)

- **HARQ:** A cyclic code is used with both detection and correction.
  - If the correction part works, there is no need to retransmit.
  - If the detection part discovers an error, and then retransmission occurs.
  - Reed Solomon cyclic codes can split the parity bytes into those for correction and those for detection (sum is the allowed maximum P).

- HARQ with **soft decoding**
  - **Chase Decoding** – use all instances of (re-) transmitted codeword (form of diversity) to decode.
  - **Incremental Redundancy** – only retransmit additional parity bits (this is what 5G uses).

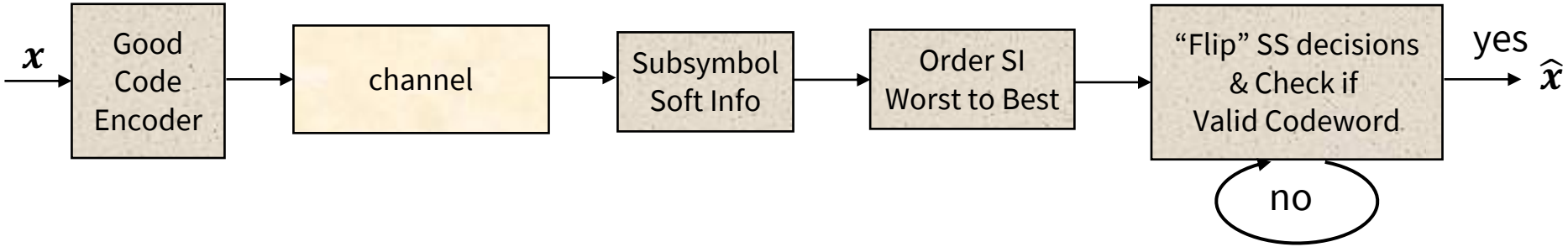
GRAND

Medard (MIT) & Duffy (Northeastern)
https://www.granddecoder.mit.edu

Eventually Section 7.6.1

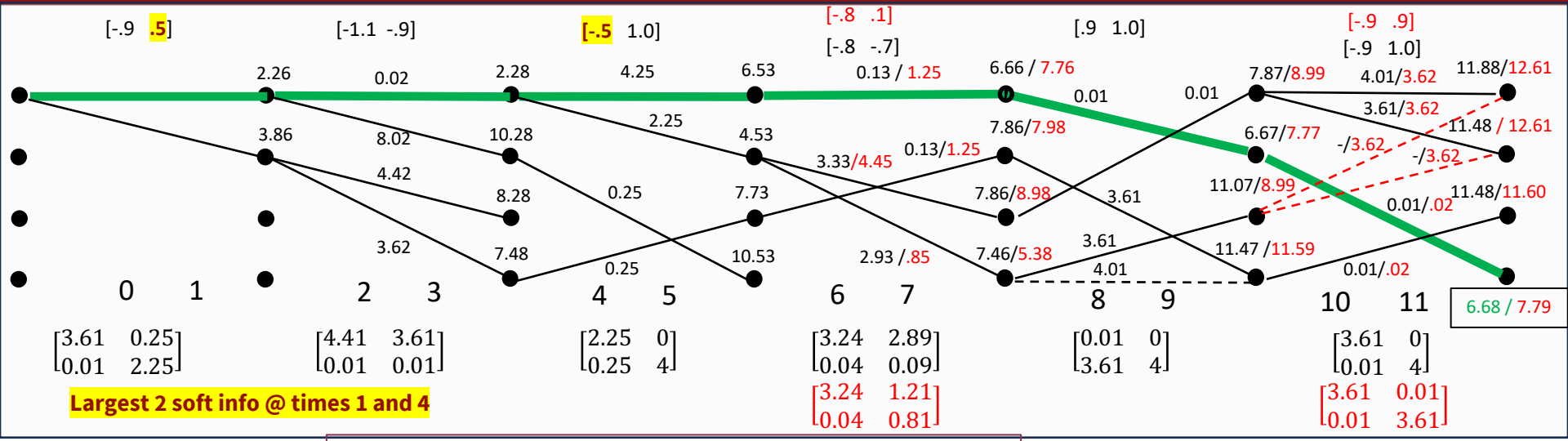
Also, see Liang (our Ethan!) and Yang work
Globecom 2019 [1],[2] –Serial-List Viterbi Algorithm

Intelligent guessing of codewords



- If (sub) symbol-by-symbol-sequence guessed decision is not a codeword already (**stop if codeword**):
  - GRAND **reorder**s the subsymbols (bits) in terms of likelihood and
  - **Guess**(es) other ss value(s) – usually means flipping n_g bit decision(s) – for least reliable LLRs.
  - Each failed guess not revisited, and it's possible (but not likely) to have up to $|C|^n$ guesses, more like $|C|^{n-k}$
    - Grows as $\binom{n}{n_{flipped\ ss}}$ guesses at each stage, usually $g = 0, \ldots, \rho$ for good codes because codewords not more than $\rho = n - k$ bits apart.
    - Expected number of steps to find codeword is $\mathbb{E}[g] = 2^{\rho}$.
    - Ensure any "give-up" point occurs less frequently than the target P_e .
- With the ss reordering from worst to best, the **first codeword found is the ML decision (low SNR)**.

Return to our L8 4-state example (stops 12 dim)



Sort soft information – ordered reliability bits (ORB)

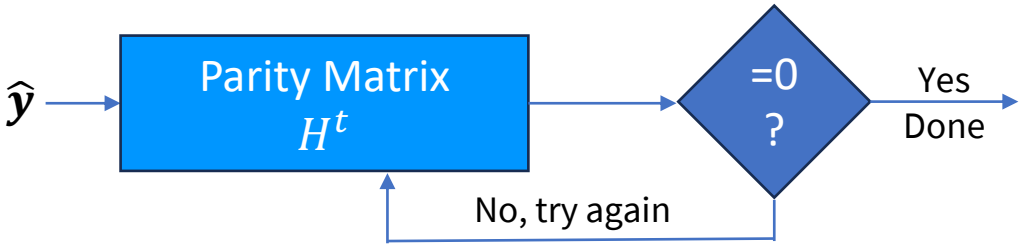
Time	1	4	7	6	0	3	8	10	5	9	11	2
Guess 0	0	0	0	0	0	0	1	0	0	1	1	0
Guess 1	0	1	0	0	0	0	1	0	0	1	1	0
Guess 2	0	0	1	0	0	0	1	0	0	1	1	0
.. Guess 11												1
Guess 12	0	1	0	0	0	0	1	0	0	1	1	0

$$\underbrace{\binom{12}{0}}_{1} + \underbrace{\binom{12}{1}}_{12} + \underbrace{\binom{12}{2}}_{66}$$

12 guesses for black case
79 guesses for red case
Times 1 & 5

Is it a codeword?

- For the example, choosing 2 bits from 12 eventually picks times 1 and 5, which is a codeword if both flipped.



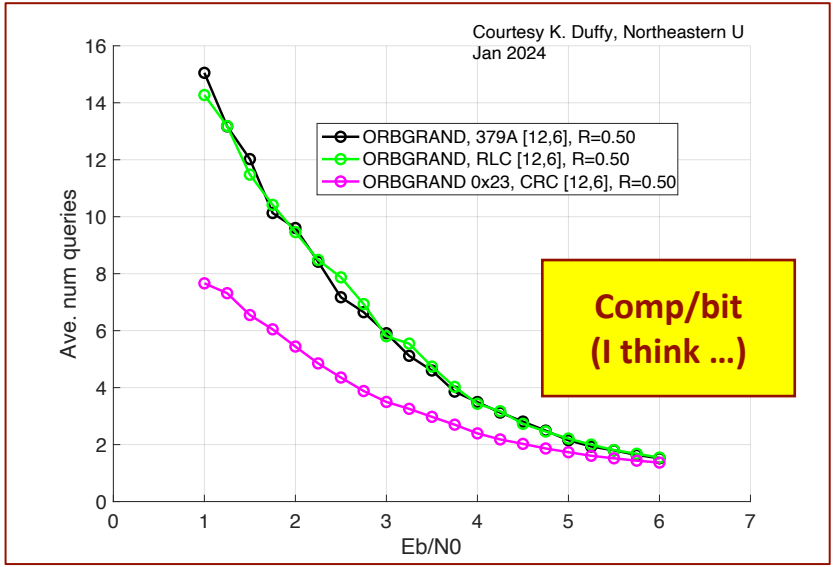
$$\underbrace{[1+D^2 \quad 1+D+D^2]}_{H(D)}$$

- For the example $H^t = \left[\underbrace{[1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]}_{h_2(D)} \underbrace{[1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1]}_{h_1(D)}\right]$, which checks (split 2 and 1).
-
- Usually, the parity check is the least complex way to check for valid codewords (linear code).
- Check after each guess to determine if ML estimate (which is first codeword to pass, given order).
  - May continue also to create a list of codewords that are next-to-ML for some soft-decoding schemes.

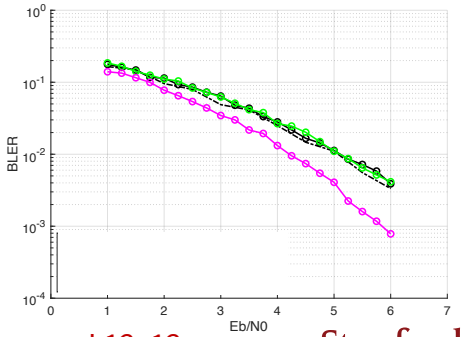
How many queries? → ORB (ordered reliability bits)

- Upper bound is 2^n.
- The guess-sequence strategy is important with good code.
- Singleton bound suggests more like 2^{n-k} for good codes [1].
  - Implementation has finite queries before exceeding compute budget.

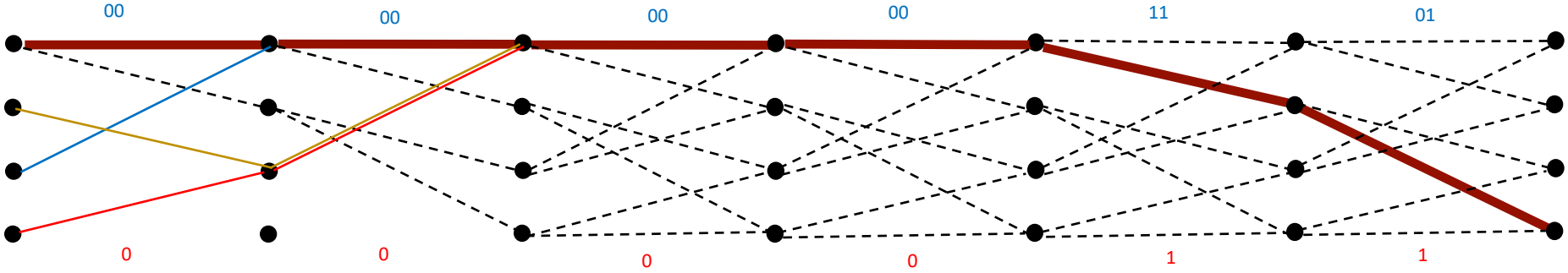
- GRAND example's computation is largely the **sort** $\leq {}^{N^2}/_{2}$
  - Finding (next) largest/smallest value is $n - i \;\; (i = 1, \ldots)$ compares
  - So 11 + 10 + 9 = [21 30] for black/red to find the ML codeword (the parity checks are trivial and number 13 and 79).
  - Viterbi is 6 x 4 x 3 = 72, roughly 2x more operations to find the same codeword (it searches unlikely paths unnecessarily).
  - So there are (12,6) codes that find ML sooner, **on average.**
  - This curve was generated with MIT software (later).



- |C| > 2 vastly increases guessing numbers, so GRAND compares better with binary codes.
- GRAND originators say correlated (nonwhite) noise (or equivalently ISI) is also well handled.
  - Not clear what convergence looks like because the number of levels per subsymbol rises rapidly with ISI.
  - As we'll see in next section of 379A and also B, there are better ways to make an ISI or correlated noise channel look like equivalent set of AWGNs to which usual good codes apply in usual manner.
  - Instructor presently believes these better match GRAND's strengths.



GRAND with BSC?



Time1	0	1	2	3	4	5	6	7	8	9	10	11
Guess 0	0	1	0	0	0	1	0	0	1	1	0	1
Guesses 1-12 (single flips)	0	1	1 (flips Move) →	0	0	0	1	0	0	0	1	1
Guesses 13-67 (two flips)	0	0	0	0	0	0	0	0	1	1	0	1

- Follows same order really – just the soft information is equal for each output bit.
- Thus, might have to go to guess 67 to get it right if unlucky (or 13 if lucky).
- With 3-bit-error case, the search could go to 299 checks = $\binom{n}{1} + \binom{n}{1} + \binom{n}{2} + \binom{n}{3}$ with n=12
  - Sort is somewhat trivial for BSC, 11+10+9 = 30 compares

MIT Matlab Software (educational uses)

Eventually Section 7.6.2

MIT programs

- Web site https://github.com/kenrduffy/GRAND-MATLAB

```
function [y_decoded,err_vec,n_guesses,abandoned] = bin_ORBGRAND1(H,max_query,y_soft)
 1-line ORGBRAND (soft detection)

 From
 K. R. Duffy, W. An, and M. Medard, "Ordered reliability bits guessing
 random additive noise decoding," IEEE Trans. Signal Process., vol. 70,
 pp. 4528-4542, 2022.
 Uses the Landslide algorithm from that paper and the light-weight
 1-line implementation introduced in
 K. Galligan, M. Médard, K. R. Duffy, "Block turbo decoding with ORBGRAND"
 arXiv:2207.11149, 2022.

 1-line ORBGRAND differs from Basic ORBGRAND by considering a dynamically
 determined intercept in its statistical model.

 Inputs:
  n          - code length
  H          - Parity check matrix or CRC function
  max_query      - Maximum number of code-book queries to abandonment
  y_soft         - Channel soft information

 Outputs:
  y_decoded      - Decoded ML codeword
  err_vec        - Putative noise
  n_guesses      - Number of guesses performed
  abandoned      - 1 if abandoned, 0 if found a codeword
```

- Calls
  - Landslide.m
- bin_ORBGRAND.m
  - Different guessing order
- bin_GRAND.m
  - Hard-coded channel output

Our Example with this software?

- 6 subsymbols and terminate, same $H(D) = [1 + D^2 \quad 1 + D + D^2]$ has the block-code H

```
H =[
   1   1   0   0   0   0   0   0   0   0   0   0
   0   1   1   1   0   0   0   0   0   0   0   0
   1   1   0   1   1   1   0   0   0   0   0   0
   0   0   1   1   0   1   1   1   0   0   0   0
   0   0   0   0   1   1   0   1   1   1   0   0
   0   0   0   0   0   0   1   1   0   1   1   1 ];
ysoft=[
  -0.9000   0.5000  -1.1000  -0.9000  -0.5000   1.0000  -0.8000  -0.7000   0.9000   1.0000  -
0.9000   1.0000];
>> [y_decoded,err_vec,n_guesses,abandoned] = bin_ORBGRAND1(H,30,-ysoft)

y_decoded =   0   0   0   0   0   0   0   0   1   1   0   1
err_vec =     0   1   0   0   0   1   0   0   0   0   0   0
n_guesses =  30 % This is more than 12, but less than 79
abandoned =   0
>> [y_decoded,err_vec,n_guesses,abandoned] = bin_ORBGRAND1(H,10,-ysoft);
abandoned =   1
ysoft =
  -0.9000   0.5000  -1.1000  -0.9000  -0.5000   1.0000  -0.8000   0.1000   0.9000   1.0000  -
0.9000   0.9000
[y_decoded,err_vec,n_guesses,abandoned] = bin_ORBGRAND1(H,300,-ysoft)

y_decoded =   0   0   0   0   0   0   0   0   1   1   0   1
err_vec =     0   1   0   0   0   1   0   1   0   0   0   0
n_guesses =  77
abandoned =   0
```

Recall that LLRi is

$$LLR_i = -\frac{2}{\sigma^2} \cdot y_i$$

- Review of guessing:
  - It tries all single errors first, and
  - then pairs of errors, etc.
- There are up to $\binom{12}{2} = 66$ patterns
  - Difference between my 13 and this 30 is ties
  - I resolved the ties favorably to my decoder's specific out.
- Need to reverse by G^{-1} to get original input bits
  - The $G(D)$ is not simply transposing $H(D)$ because of the block termination.
- For this case, we can look at trellis
  - Presumes starts in state 00.
- More generally, the program does not appear to provide the input estimate, so appears to presume systematic – or that G^{-1} is available.
- Did not need 299 guesses, only 77 for 3-error case.

With Hard Decoded

- Bin_GRAND.m

```
function [y_decoded,putative_noise,n_guesses,abandoned] =
bin_GRAND(H,max_query,y_demod)
  GRAND (hard detection, BSC)

 Inputs:
   n           - code length
   H           - Parity check matrix or CRC function
   max_query   - Maximum number of code-book queries to abandonment
   y_demod     - Channel hard output

 Outputs:
   y_decoded     - Decoded codeword
   putative_noise  - noise
   n_guesses     - Number of guesses performed
   abandoned     - 1 if abandoned, 0 if found a codeword
```

```
>> y=[0 1 0 0 0 1 0 0 1 1 0 1]

>> [y_decoded,err_vec,n_guesses,abandoned] = bin_GRAND(H,30,y)
y_decoded =
   0   0   0   0   0   0   0   0   1   1   0   1
err_vec =
   0   1   0   0   0   1   0   0   0   0   0   0
n_guesses =   28
abandoned =    0
>> y =[   0   1   0   0   0   1   0   1   1   1   0   1];
>> [y_decoded,err_vec,n_guesses,abandoned] = bin_GRAND(H,300,y)

y_decoded =    1   1   0   1   0   1   1   1   1   1   0   1
err_vec =      1   0   0   1   0   0   1   0   0   0   0   0
n_guesses =  101
abandoned =    0
```

- The query count differs slightly because the hard-decoded example has different LLR inputs.

GRAND Soft Information

Eventually Section 7.6.3

An incorrect codeword guess

- ML codeword is not always correct – so GRAND may also guess this incorrect codeword.
- This is an error.
- A guess $\hat{\boldsymbol{x}}$ is correct with probability (from Chapter 1), call this guess q^*.

$$p_{\boldsymbol{x}/\boldsymbol{\Delta}}(\hat{\boldsymbol{x}}/\boldsymbol{\delta}) \text{ , where } \boldsymbol{\Delta} \triangleq [|y_1 - \hat{x}_1|^2 \quad \cdots \quad |y_n - \hat{x}_n|^2] \textbf{ with/without reordering}$$

- Current/past guesses' probability, when subtracted from 1, is soft information.
  - This is, possibly with scaling (seq to code), proportional to the probability that the ML $\hat{\boldsymbol{x}}$ is incorrect.
  - For guess q , the intrinsic input probability will be $p_{q,i}$ for $i = 1, \ldots, n$, initially this is $p_{0,int,i} = 2^{-1}$.

$$p_{q,int,i} = \frac{e^{LLR_i}}{1+e^{LLR_i}} \qquad p_{q,chan,i} = \frac{e^{-\frac{\|\Delta_{q,i}\|^2}{2\sigma^2}}}{(\sqrt{2\pi\sigma^2})} \qquad p_{q,i} = p_{q,chan,i} \cdot p_{q,int,i}$$

- Probability of channel-output (noise seq) guess q

$$P(\boldsymbol{n}_q) = \prod_{i=1}^{n} \left(p_{q,i\in\{n_{q,0}\}}\right) \cdot \left(1 - p_{q,i\in\{n_{q,1}\}}\right)$$

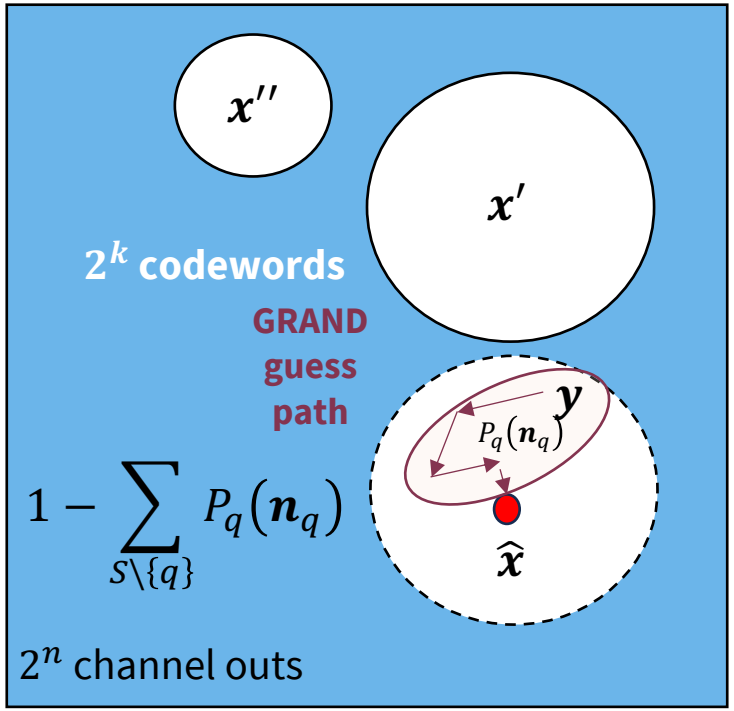
seq has 0 @ position i — seq has 1 @ position i

- Probability guesses **after** q^*

$$P(A) = 1 - \underbrace{\sum_{q=0}^{q^*} P_q(\boldsymbol{n}_q)}_{before\ and\ q^*}$$

AEP-like GRAND Analysis

- 2^n possible $\boldsymbol{y}$ (noise) guesses, S:
  - Continuous to discrete dist'n
- Grand guesses' path eliminates q^* of these guesses.
- So, $1 - \sum_{S\setminus\{q\}} P_q(\boldsymbol{n}_q)$ is the probability of the other outputs not yet guessed, which is $\propto$ probability of ML error.
- 2^k possible $\hat{\boldsymbol{x}}$ guesses
  - On average, each associates with fraction $\frac{2^k-1}{2^n-q^*}$ $= \frac{\#\ of\ (other)\ codewords}{\#\ of\ remaining\ guesses}$



$$P(A) \triangleq \left(1 - \sum_{q=0}^{q^*} P(\boldsymbol{n}_q)\right) \cdot \underbrace{\frac{2^k - 1}{2^n - q^*}}_{\text{scale for } codewords}$$

- The sequence likelihood of correct is
  - $\frac{P(\boldsymbol{n}_{q*})}{P(A)+P(\boldsymbol{n}_{q*})}$
- While incorrect is
  - $\frac{P(A)}{P(A)+P(\boldsymbol{n}_{q*})}$

The ML codeword $\hat{\boldsymbol{x}}$ and set of all others $S_{\boldsymbol{x}\setminus\hat{\boldsymbol{x}}}$ are mutually exclusive, so can add their probabilities.

Bit-level Likelihood

- When $\hat{\boldsymbol{x}}_i = 0$ for guess q^*:
  - $LLR_i = \frac{P(\boldsymbol{n}_{q*}) + P(A) \cdot p_{q^*,i}}{P(A) \cdot (1 - p_{q^*,i})}$

$$\frac{seq\ prob + 0\ anyway\ with\ others}{1\ with\ others}$$

- When $\hat{\boldsymbol{x}}_i = 1$ for guess q^*:
  - $LLR_i = \frac{P(A) \cdot p_{q^*,i}}{P(\boldsymbol{n}_{q*}) + P(A) \cdot (1 - p_{q^*,i})}$

$$\frac{0\ with\ others}{seq\ prob + 1\ anyway\ with\ others}$$

- These provide individual bit decision's soft information based on the codeword likelihood.
- These use the reordered $\boldsymbol{\Delta}_q$'s .

But what about Ext? Int?

- After the above calculations are complete,
  - $LL_{ext,i} = LL_i - log(p_{q,chan,i}) - log(p_{q,int,i})$ - where $p_{q,i} = p_{q,chan,i} \cdot p_{q,int,i}$ is time i only ss prob.
  - Where the intrinsic is the a priori or previous decoder's supplied extrinsic

A GRAND Iterative Decoder

Examples?

- I don’t have one yet.
- There is no MIT publicly available software to compute this soft information.
- It appears tedious, but probably not that much calculation in that the successive guesses only differ in one or a few bit positions – so incrementally updated at only the guess-bit positions.
- The inventors claim very significant decoding complexity reduction, on average.
  - And it appears they are correct.

Great extra credit problem for motivated student,

But don’t underestimate this either.

Product Codes & GRAND

Section 8.3.3

Product Codes (PCs) in General

- Begin with Example (from J. Gill's EE387)
  - Two (8,4,4) expanded Hamming codes:

Code1 / Code 2	$k_2 = 4$	$n_2 - k_2 = 4$
$k_1 = 4$	Input data bits (16)	Parity for code 2 (16)
$n_1 - k_1 = 4$	Parity for code 1 (16)	Parity on parity (16)

- There are $k_1 \cdot k_2$ input bits (so 16).
- There are $n_1 \cdot n_2$ output bits (so 64).
- The rate is $r = r_1 \cdot r_2$ (so 1/4).
- Free distance is $d_{free,1} \cdot d_{free,2}$ (so 16).
- This product code corrects up to 7 errors.

Larger distance, lower rate

For AWGN, the ML decoder is difficult (like most serious block codes)

There are also product code of convolutional codes; this is turbo without the random interle

GRAND offers PC an iterative decoding solution

- GRAND Code 1 (horizontally) – compute soft information (previous section).
- GRAND Code 2 (vertically) -- use Code 1's extrinsic information as intrinsic input
- Repeat the cycle
  - (16,11)~Exp Hamming

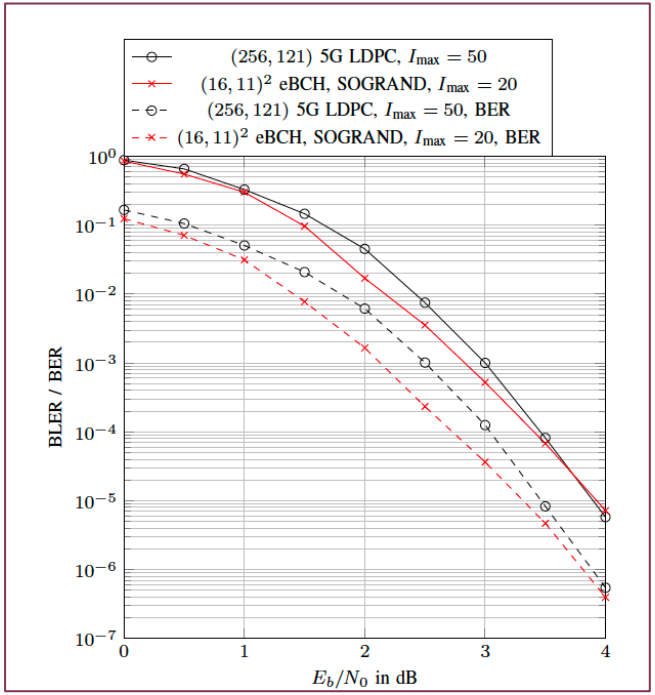
Soft-output (SO) GRAND and long, low rate codes to outperform 5 LDPCs.

Peihong Yuan and Muriel Médard
Research Laboratory for Electronics
Massachusetts Institute of Technology
Cambridge, USA
{phyuan,medard}@mit.edu

Kevin Galligan
Hamilton Institute
Maynooth University, Ireland
kevin.galligan.2020@mumail.ie

Ken R. Duffy
Dept. of ECE & Dept. Mathematics
Northeastern University
Boston, USA
k.duffy@northeastern.edu

arXiv:2310.10737v3 [cs.IT] 4 Dec 2023



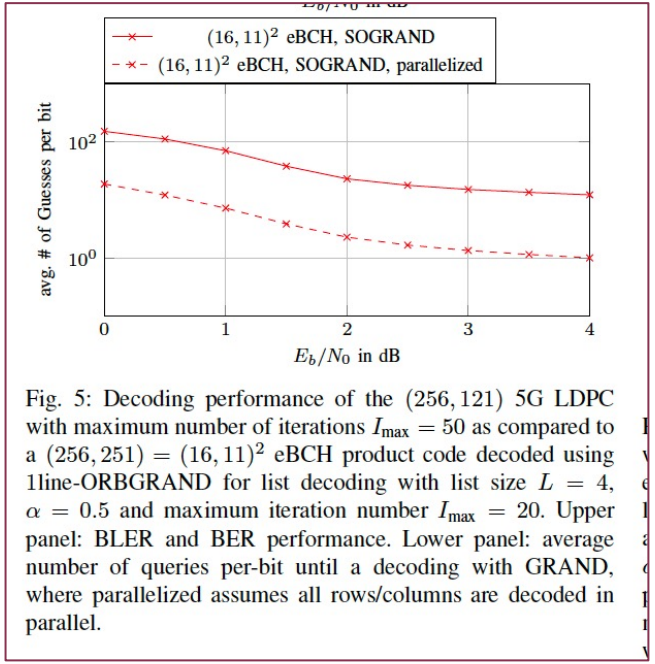


Fig. 5: Decoding performance of the $(256, 121)$ 5G LDPC with maximum number of iterations $I_{\max} = 50$ as compared to a $(256, 251) = (16, 11)^2$ eBCH product code decoded using 1line-ORBGRAND for list decoding with list size $L = 4$, $\alpha = 0.5$ and maximum iteration number $I_{\max} = 20$. Upper panel: BLER and BER performance. Lower panel: average number of queries per-bit until a decoding with GRAND, where parallelized assumes all rows/columns are decoded in parallel.

It's not clear (to met yet) exactly how the 50 iterations LDPC compares to GRAND's 20

Seems to be best codes yet (full circle to beginning)

- It's the decoder. Most codes are good, just AWGN-ML decoder previously appeared too complex.
- via GRAND – perhaps ML decoding is not too complex after all on the simple product codes.
- Puncturing and rate-adaptation?
  - The product codes need not be the same for horizontal and vertical.
  - This creates a lot of flexibility – perhaps more than puncturing (so punt the puncture? 😀)
- High-speed parallel implementations
  - Active research area.

Now, we have a few good methods for $\Gamma \rightarrow 0$ dB
&
Can exploit them to optimize larger gains from other effects!

Comparing Ethan's codes to ORBGRAND

- It's the decoder. Most codes are good, just AWGN-ML decoder previously appeared too complex.
- via GRAND – perhaps ML decoding is not too complex.

$$\frac{E_b}{N_0} = R \cdot \mathrm{SNR} \Rightarrow \mathrm{SNR} = \frac{1}{\mathrm{R}} \frac{\mathrm{E_b}}{\mathrm{N_0}}$$

- Consider operating with SNR = 3dB:
  - The previous ORBGRAND plot operates at r < ½, which implies their $\frac{\mathrm{E_b}}{\mathrm{N_0}} = 3$ dB corresponds to a better channel than the 3dB SNR in my channel. (There's a factor of 2 scaling somewhere).
  - The list-decoded convolutional code outperforms ORBGRAND in terms of FER at higher code rate.

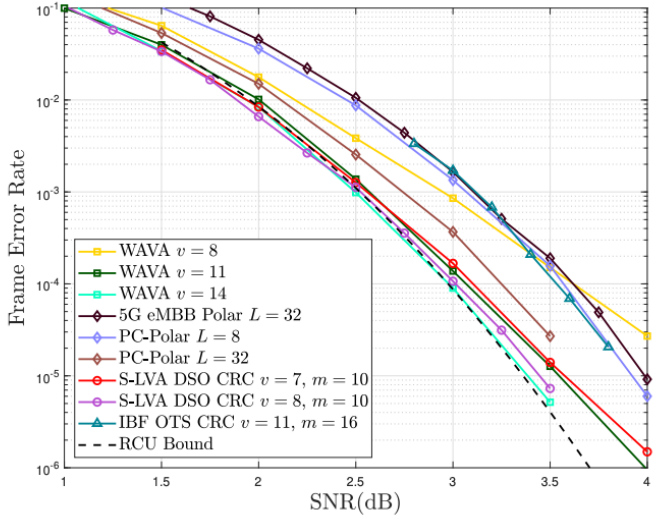


Fig. 6. FER vs. SNR comparison between serial-list Viterbi decoded TBCC with DSO CRC and other candidate codes for URLLC. All simulations are for $k = 64$, $n = 128$, rate=1/2. WAVA and 5G eMBB simulations are located in [26], parity-check polar (PC-Polar) code simulations are located in [7], and iterative bit flipping (IBF) simulations with an off-the-shelf (OTS) CRC are located in [8].

End Lecture 12

[1] Ethan Liang , Hengjie Yang , Dariush Divsalary, and **Richard D. Wesel, "**List-Decoded Tail-Biting Convolutional Codes with Distance-Spectrum Optimal CRCs for 5G*," IEEE Globecom 2019.*

[2] Hengjie Yang , **Ethan Liang** , Hanwen Yaoy, Alexander Vardyy, Dariush Divsalarz, and **Richard D. Wesel ,** A List-Decoding Approach to Low-Complexity Soft Maximum-Likelihood Decoding of Cyclic Codes, Globecom 2019.