Supplementary Lecture 10B

ORBGRANDn & SOGRAND Iteration

February 10, 2026

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

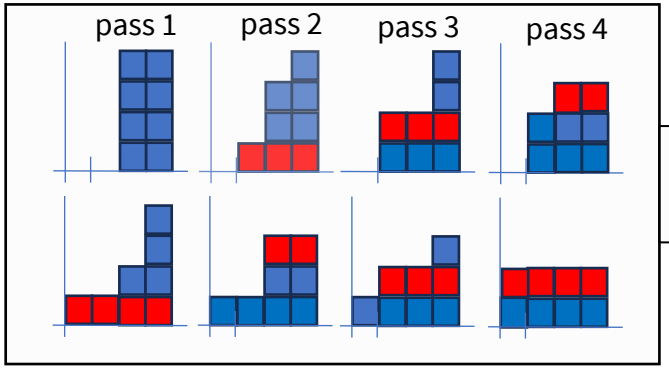
Instructor EE379A – Winter 2026

Agenda

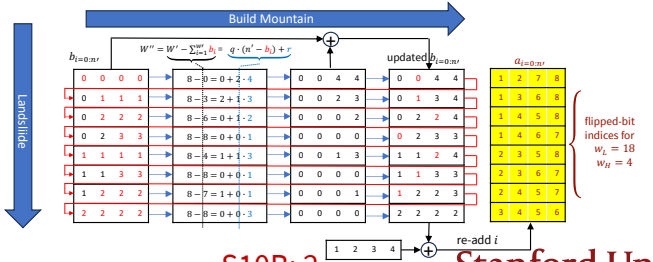
- S10B Topics
  - Rate – ½ Example (follows L10 soft-output generation on slides L10:31;33) with ORBGRAND1
  - Piecewise Linear Extension (ORBGRANDn)
  - Iterative GRAND software

The rapid landslide GRAND search has 2 exploitations:
Extends to piecewise linear SNR approximation,
Allows iterative decoding (soft output information generation)

Verbal error in S10A:
Here correct to always increases by 1 in the position of max drop



$$n = 8\,, w_H = 4\ ;$$
$$w_L = 18; w_L' = w_L - 5 \cdot 4/2 = 8\ ;$$
$$n' = n - w_H = 4$$



Rate – ½ (6/12) CC example (ORBGRAND1)

Section 7.2.3.2

SOGRAND's Probability of Codeword Found

- GRAND decision (codeword) is not necessarily MAP/ML, nor is it always correct.
  - q^* is 1st decoder-decision index when $\left[\hat{\boldsymbol{n}}_{q^*} \oplus \frac{hard\left(\hat{\boldsymbol{y}}_{q^*}\right)+1}{2}\right] \in C$
  - where "hard" rounds to ± 1, and $\hat{\boldsymbol{n}}_q \leftrightarrow \hat{\boldsymbol{x}}_q$, which is 1-to-1 **for** codewords when found , THUS:

- SOGRAND for $q = 0, \ldots, q^*$, accumulates $\sum_{q=0}^{q^*} P(\boldsymbol{n}_q)$.
  - The probability a decision $\hat{\boldsymbol{v}}$ (codeword) remains not found is

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

- Each (output) channel bit position has $LL_{q,i} = -\frac{ln(2\pi\cdot\sigma^2)}{2} - \frac{\left(y_i - [2\cdot v_{q,i}-1]\right)^2}{2\cdot\sigma^2} - \ln(p_i)$ or $LLR_i = -\frac{2\cdot y_i}{\sigma^2}$.
  - If there is an $\grave{a}$ priori (intrinsic) for position i with $LL_{ap,i}(v_i)$, then this should replace the $-\ln(p_i)$ above.
- The guess-sequence probabilities are each $P(\boldsymbol{n}_q) = \sum_{i=1}^{n} e^{LL_{q,i}}$, and then they can be summed over q.

Easier way is to compute $LL_{\mathbf{0},i}$ and then just compute/add w_q correction terms for each guess

Rate ½ cc Example

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

H= [1100000000000
0111000000000
1101110000000
0011011100000
0000110111000
0000001101111];

- First compute the 12 values of $LL_{q,i} = -\frac{ln(2\pi\cdot\sigma^2)}{2} - \frac{(y_i-[2\cdot v_{q,i}-1])^2}{2\cdot\sigma^2} + \ln(p_i)$ for

$y_{1:12}$ = -0.9000 0.5000 -1.1000 -0.9000 -0.5000 1.0000 -0.8000 -0.7000 0.9000 1.0000 -0.9000 1.0000];

  - Add their magnitudes for $LL_{\mathbf{0}}$.
  - For each subsequent guess, subtract $|LL_{0,i}|$ and add $-\frac{ln(2\pi\cdot\sigma^2)}{2} - \frac{(y_i-[2\cdot v_{q,i}-1])^2}{2\cdot\sigma^2} + \ln(p_i)$ in the $w_{H,q}$ positions that differ from all zeros to get LL_q.
  - Then iteratively compute

$$\sum_{q=0}^{q^*} P(\boldsymbol{n}_q)$$

For the r=1/2 convolutional code, see L10 simplified calculation

- Proceed with calculations on L10:33 for individual soft-bit outputs and compute p'_{nf} $(= \frac{2^k-1}{2^{n-q^*}} \cdot p_{nf})$.
  - Non-initial iterations remove intrinsic components, $LL_{ext,i} = LLR_i - LLR_{q,chan,i} - LLR_{q,int,i}$, before passing to another GRAND.

More detail on $1 - p_{nf} = \sum_{q=0}^{q^*} P(\boldsymbol{n}_q)$ calculation

- Recall from S10A:6
- $N_i \triangleq$ number of flips of bit i at/before quess q^*.

Guess	|LLR| sum	[dimension indices]
1	0.00000000	[ ]
2	1.79662235	[2]
3	1.79662235	[5]
4	2.51527129	[8]
5	2.87459576	[7]
6	3.23392023	[1]
7	3.23392023	[4]
8	3.23392023	[9]
9	3.23392023	[11]
10	3.59324470	[2 5]
11	3.59324470	[6]
12	3.59324470	[10]
13	3.59324470	[12]
14	3.95256917	[3]
15	4.31189364	[2 8]
16	4.31189364	[5 8]
17	4.67121811	[2 7]
18	4.67121811	[5 7]
19	5.03054258	[1 2]
20	5.03054258	[2 4]
21	5.03054258	[1 5]
22	5.03054258	[4 5]
23	5.03054258	[2 9]
24	5.03054258	[5 9]
25	5.03054258	[2 11]
26	5.03054258	[5 11]
27	5.38986705	[2 6]} $q^* + 1 = 27$
28	5.38986705	[5 6]
29	5.38986705	[7 8]
30	5.38986705	[2 10]

Bit i	N_i
1	3
2	8
3	1
4	3
5	8
6	1
7	3
8	3
9	3
10	1
11	3
12	1

1. The LL is $LL_{q,i} = -\frac{ln(2\pi\cdot\sigma^2)}{2} - \frac{(y_i - [2\cdot v_{q,i} - 1])^2}{2\cdot\sigma^2} + \ln(p_i)$
2. $LL_{q^*,i}(1) = -\frac{ln(2\pi\cdot\sigma^2)}{2} - \frac{(y_i-1)^2}{2\cdot\sigma^2} + \ln(1-p_i)$ for N_i occurrences in $\sum_{q=0}^{q^*} P(\boldsymbol{n}_q)$
3. And $LL_{q^*,i}(0) = -\frac{ln(2\pi\cdot\sigma^2)}{2} - \frac{(y_i+1)^2}{2\cdot\sigma^2} + \ln(p_i)$ for $(2^{q^*+1} - N_i)$ occurences in $\sum_{q=0}^{q^*} P(\boldsymbol{n}_q)$
4. Compute $LL_{\mathbf{0}} = \sum_{i=1}^{n=12} LL_{0,i}$
5. Then $1 - p_f = \sum_{q=0}^{q^*} P(\boldsymbol{n}_q) = 2^{q^*+1} \cdot LL_{\mathbf{0}} - \sum_{i=1}^{n=12} (2^{q^*+1} - N_i) \cdot LLR_{0,i}$

Design might use recursive implementation of step 5.

Piecewise Linear Extension (ORBGRANDn)

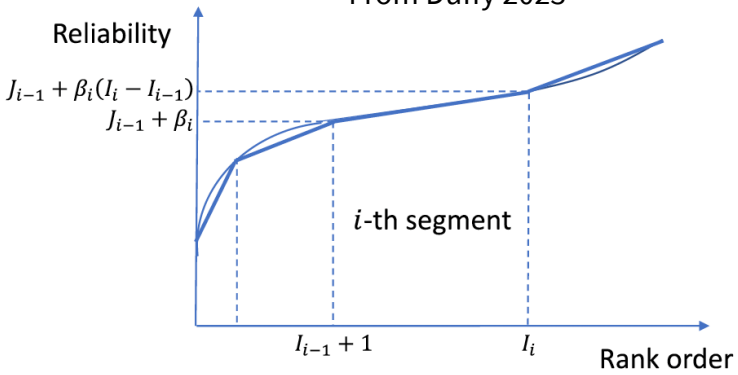
Section 7.1.2

Piecewise Linear Fits in ORBGRAND

- ORBRANDn uses Landslide in $n+1$ segments.
- The n choice depends on the LLR_i data
  - $n \leq 3$ in practice.
  - Segment endpoints are (I_i , J_i).
  - The m sets are:

$$\Xi_i = \left\{ \underbrace{[w_{L,1}, \ldots, w_{L,m}]}_{\boldsymbol{w}_L} \in (\mathbb{Z}^+)^m \mid \sum_{i=1}^{m} w_{L,i} = w_L \right\}$$

From Duffy 2023



- Scaling ensures integers everywhere, so scale all $\{LLR_i\}$ by Q:

$$Q = \min\left\{ \frac{LLR_{I_1} - LLR_1}{I_1 - 1}, \min_{i \in [2,m]} \left[\frac{LLR_i - LLR_{i-1}}{I_i - I_{i-1}} \right] \right\}$$

$$\begin{aligned}
\beta_1 &= \left\lfloor \frac{LLR_{I_1} - LLR_1}{(I_1 - 1) \cdot Q} \right\rceil , i = 1 \\
\beta_i &= \left\lfloor \frac{LLR_{I_i} - LLR_{i-1}}{(I_i - I_{i-1}) \cdot Q} \right\rceil , i = 2 : m \\
J_0 &= \left\lfloor \frac{LLR_{i-1}}{Q} \right\rceil - \beta_1 , \; i = 0 \\
J_i &= \left\lfloor \frac{LLR_i}{Q} \right\rceil , \; i = 1 : m-1
\end{aligned}$$

$$\widehat{LLR}_j = \beta_i \cdot (j - I_{i-1}) + J_{i-1} \; , \; \forall \; I_{i-1} < j \leq I_i \; , \; i = 0 : m \wedge \; j = 1 : n \; .$$

Adjusting Algorithms for each segment

- Each segment has logistic-weight increment,
  - which is integer using Q on previous slide

$$\delta w_{L,i} = \frac{w_{L,i} - w_{H,i} \cdot J_{i-1}}{\beta_i}$$

- This is the part to which the segment's landslide/mountain-build is applied for the given $w_{L,i}$.
- But for each w_L , what are the individual values $w_{L,i}$?
- This requires a second level of integer partitioning (next slide on integer splitting).

All the segments' logistic weight must add to total.

- Integer-splitting enumerates the possible $w_{L,i}$ partitions for each of the m segments that add to w_L.
- **Integer Splitting** (another integer partitioning problem – find them all)

Inputs: $\boldsymbol{w}_L$ and m
Outputs: $\{\boldsymbol{w}_L^{m,k}\}$

Integer Splitting Algorithm

1. $\boldsymbol{w}_L = \emptyset$
2. $k = 0$
3. for $w_{L,1} = 0 : w_L$
   (a) **for** $w_{L,2} = 0 : w_L - w_{L,1}$
      i. $\ddots$
         for $w_{L,m} = 0 : w_L - \sum_{i=1}^{m-1} w_{L,i}$
            $\boldsymbol{w}_L^{m,k} = \{\boldsymbol{w}_L\ , [w_{L,1} \ldots w_{L,m}]\}$
         endfor
   endfor
endfor

- **Collection Constraints**
  - Discard $w_L^{m,k}$ that do not meet these constraints:

$$
\begin{aligned}
w_{L,i} &= 0 \text{ or } w_{L,i} - w_{H,i} \cdot J_{i-1} \geq \frac{w_{H,i} \cdot (w_{H,i}+1)}{2} \\
w_{L,i} - w_{H,i} \cdot J_{i-1} &\leq (I_i - I_{i-1} + 1) \cdot w_{H,i} - \frac{w_{H,i} \cdot (w_{H,i}+1)}{2} \\
w_{L,i} - w_{H,i} \cdot J_{i-1} &\ / \ \beta_i \quad (\text{That is } \beta_i \text{ divides } w_{L,i} - w_{H,i} \cdot J_{i-1})
\end{aligned}
$$

- Substantially fewer guesses to test with parity check.
- More segments increase search complexity,
  - but performance approaches more closely SGRAND.

Iterative GRAND

Section 7.3.6.5

Iterative GRAND software

- Special version provided by K. Duffy – not available at github, just at course web site, sim_product.m

```
clear;
%% Monte-Carlo parameters
EbN0dB = 1.5:0.25:2.5;
NoErrors = 20;
maxIt = 10^6;
minIt = 10^2;
%% Code parameters
n = 31;
k = 25;
code_class = 'CRC';
[G, H] = getGH_sys_CRC(n, k); % CRC from Koopman's database


%% Alternative classic TPC component code suggestion
%
% n = 32;
% k = 26;
% code_class = 'eBCH';
% [G, H] = getGH_sys_eBCH(n, k); % Extended BCH


%% Decoder parameters
L = 4; % Maximum list size
Imax = 20; % maximum number of iterations
Tmax = Inf; % Maximum number of queries per componet decoding
p_ET = 1e-5;
thres = 1 - p_ET; % Abandon decoding if list already has >thres prob.
alpha = 0.5 * ones(1, 50); % Extrinsic LLR scaling
```

Currently a circular binary code
Comment these out.

insert your own binary code parameters
here with [G , H] and uncomment these in.

```
>> sim_product
---CRC^2, [961,625]---Eb/N0 dB 1.5
dB:---
 BLER         = 0.48
 BER          = 0.03024
 Iavg         = 13.865
 NGavg        = 115747.79
 NGavg/(info bit) = 185.1965
 NGavg_p      = 8365.34
---CRC^2, [961,625]---Eb/N0 dB 1.75
dB:---
 BLER         = 0.18519
 BER          = 0.01083
 Iavg         = 7.9167
 NGavg        = 67827.4537
 NGavg/(info bit) = 108.5239
 NGavg_p      = 4840.7778
---CRC^2, [961,625]---Eb/N0 dB 2 dB:--
-
 BLER         = 0.056022
 BER          = 0.0025725
 Iavg         = 4.8768
 NGavg        = 43061.4566
 NGavg/(info bit) = 68.8983
 NGavg_p      = 3069.3838
```

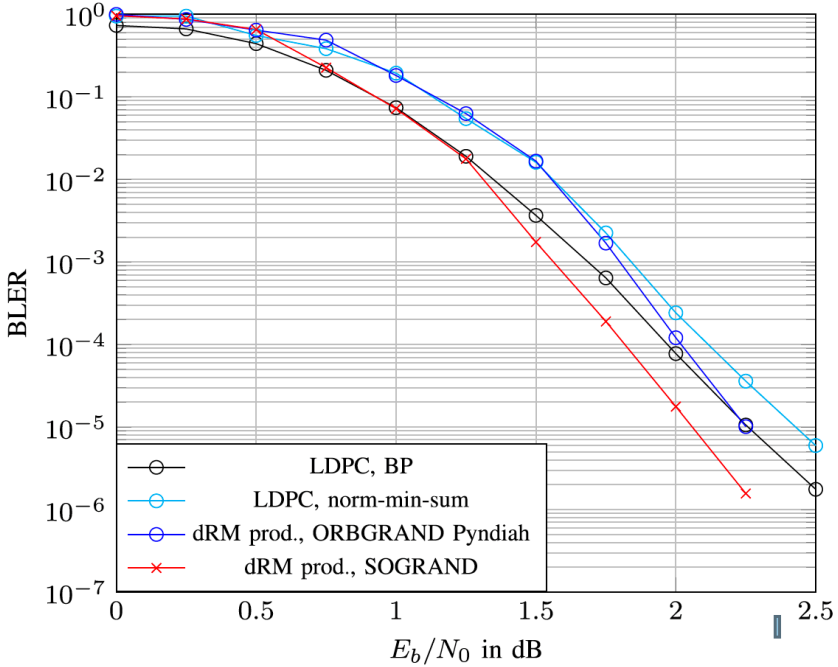
```
---CRC^2, [961,625]---Eb/N0 dB 2.25
dB:---
 BLER         = 0.0060772
 BER          = 0.00024114
 Iavg         = 3.1123
 NGavg        = 27736.5336
 NGavg/(info bit) = 44.3785
 NGavg_p      = 1931.7581
---CRC^2, [961,625]---Eb/N0 dB 2.5
dB:---
 BLER         = 0.0014484
 BER          = 3.0127e-05
 Iavg         = 2.3366
 NGavg        = 21048.873
 NGavg/(info bit) = 33.6782
 NGavg_p      = 1446.9421
```

Iavg = average iterations to converge
NGavg = ave number of guesses/iter
NGavg/(info bit) = ave guesses/bit
NGavg_p = # passes / iter

Use to produce Pe vs SNR plots

Product Code Example

- Product Code is RM (32, 21) code squared,
  - So rate is 441/1024
  - Same rate for LDPC code
- The simple product code is 1 dB better than best 5G LDPC code with GRAND.
- There remains some dispute on best code use, but instructor believes the GRAND approach is the next step in most powerful codes, using product codes of well known BCH (extended to be even) and RM.
- Beyond GRAND, only shaping gain (which is independently applied, see Chapter 8) for large constellations remains.
- Basically, we've met Shannon's AWGN promise with these simple product codes, GRAND, and BICM



As EE379AB students progress, they will see that these good single-user codes will be reusable canonically on all multiuser, filtered additive Gaussian noise channels.

End Lecture S10B