Supplementary Lecture 10A

GRAND Simplification via Landslide Algorithm

February 10, 2026

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

Instructor EE379A – Winter 2026

Agenda

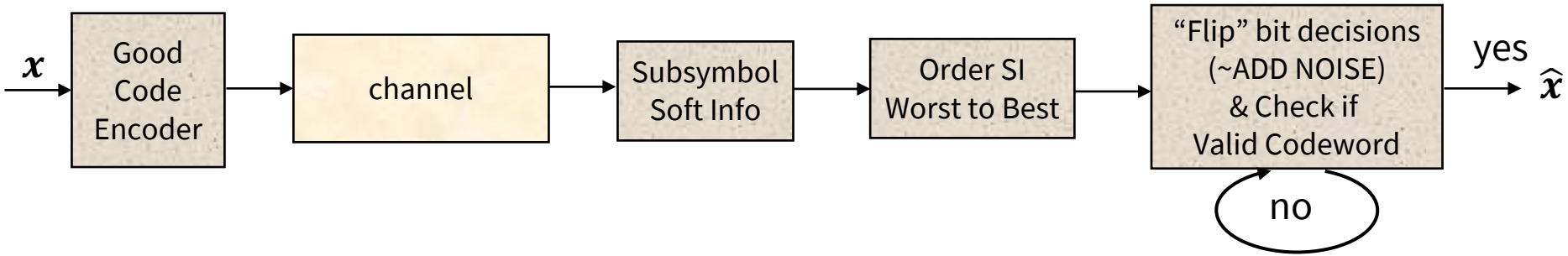
GRAND's actual search can be greatly simplified by an SNR-dependent noise-adding (bit flipping) precomputable order. (Decoder doesn't need to check all flips.)

- S10A Topics
  - GRAND simplification
  - Mountain Building and Landsliding Algorithms

GRAND Simplification

Section 7.1 2.3

GRAND – See L10



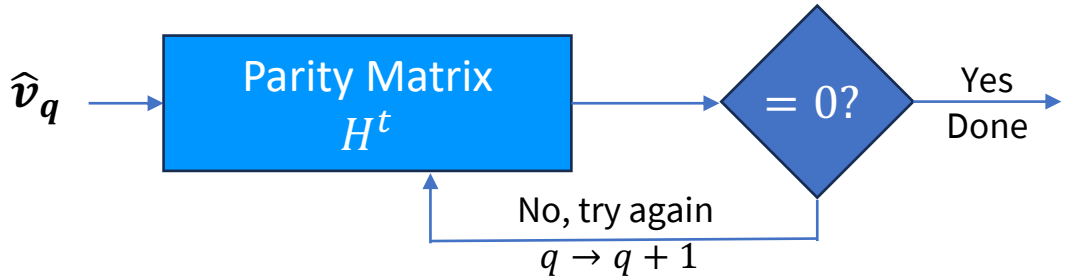
- If guessed bit-sequence decision is not a codeword already (**stop if codeword**):
  - GRAND **reorder**s the subsymbols (bits) in terms of sequence likelihoods (summed LLRs, indexed by q, "query" or "guess").
    - $q = 0$ guess is just try hard-decoded output with no added noise (no flips).
    - Smallest LLR sum tests first at $q = 1$, & then guesses $q > 1$ are tested to be codeword.
  - Each guess equivalently **flips** bit hard-decoded sequence for the q^{th} smallest |LLR| sum.

- Failed guesses are not revisited: $2^n - k + 1$ guesses are possible (not likely), the random-guess average is 2^{n-k}.
  - Ideally, GRAND solves the shortest-path problem (orders sums of possible nonnegative real numbers, |LLR|'s), so often less than 2^{n-k}.
  - The expected number of guesses to find a codeword is $\mathbb{E}[q] = 2^{n-k}$, but reduces with SNR-dependent guessing strategies that skip unlikely sum values.
  - Ensure any "abandon-search" point occurs less frequently than the target P_e.

- The **first codeword found is often the ML decision (low SNR)**, but GRAND is **not an ML** decoder.
  - Ordering LLR sums is not the same as ordering the squared-error sums.

Is it a codeword?

- For hard decoding, choosing 1, then 2, then 3 bits from 12 eventually will find a codeword $\boldsymbol{y} + \boldsymbol{n}_q = \hat{\boldsymbol{v}}_q$ where parity $\hat{\boldsymbol{v}}_q \cdot H^t = \boldsymbol{0}$.
  - The $\boldsymbol{n}_q$ noise guess is really a pattern of 0's and 1's , so really something added to a hard ($\boldsymbol{y}_q$). The LLR values are not binary.



- For example:

```
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];
```

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

- The parity check is easy part – it is the guess-ordering computation that requires more effort.

Ordered Bit Reliability Example

- Return $r = 1/2$ convolutional code example, terminates at 6 input bits (12 output dimensions)

ysoft=[-3.2339 1.7966 -3.9526 -3.2339 -1.7966 3.5932 -2.8746 -2.5153 3.2339 3.5932 -3.2339 3.5932]

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]}
28	5.38986705	[5 6]
29	5.38986705	[7 8]
30	5.38986705	[2 10]

ORBGRAND is ML decision here

ORBGRAND1 produced this same decision at $q = 30$, slightly later in this case, but without table calculations.
So how? (coming next)

Ordered Bit Reliability Simplification

- Ordered Bit Reliability (ORB → "**ORBGRAND**") simplifies GRAND when the AWGN-channel SNR is known.
  - ORB-GRAND eliminates many guesses that are unlikely to correspond to errors (SNR).

- ORBGRAND usually uses only even $\{d_{free} \in 2\mathbb{Z}^+\}$ codes.
  - Reduces search by ½ (so our earlier example would have been much less complex if $r = 6/13$ and higher performance)
    - If odd, G augments sum of other columns, $r \rightarrow k/(n+1)$ to make it an even code.
    - This also increases d_{free} by 1.

- The decoder uses an SNR-specific order-of-sums to avoid comparisons.
  - This order-of-sums (or really corresponding indices of summed LLRs) follows a (piecewise) linear (offset) LLR curve,
  - which can be approximated by a **logistic weight** (with d_i=1 if bit flipped and 0 otherwise)

$$w_H = \sum_{i=1}^{n} v_i \quad ; \quad w_L \triangleq \sum_{i=1}^{n} i \cdot v_i \quad , \text{ where } i \text{ orders the } LLR_i \text{ s from smallest to largest.}$$

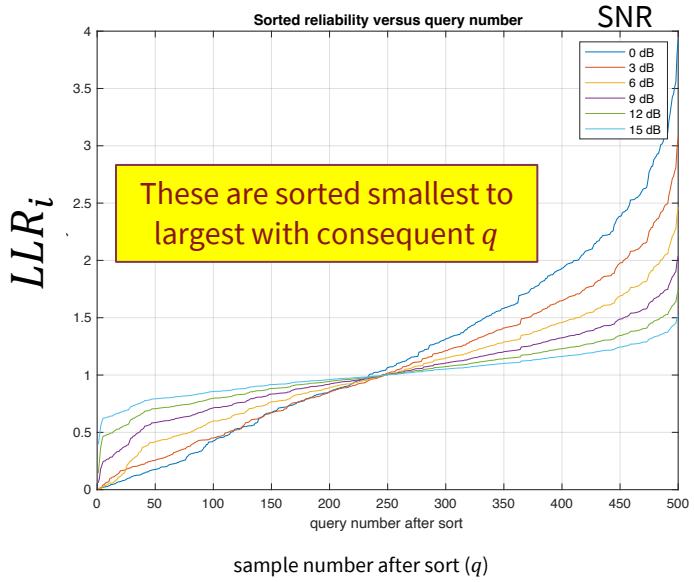
- Reliability is approx. prop to logistic weight, over SNR range (See next slide.). So, precompute the index combinations with lowest w_L .

Calculation of possible noise-flip patterns

- GRAND (theoretically) first sorts the LLR_i over $i \in 1{:}n$.
  - Noise guessing ~ orders samples from a Gaussian distribution.

- For SNR=0 dB, the $|LLR_i|$ looks linear.
  - It is rare to get to the right portion of the curve.

$$|LLR_i| = \beta \cdot i$$



- **Logistic weight** $w_L = \sum_{i=1}^{n} i \cdot v_i \quad where\ v_i \in \{0,1\}$
  - Hamming weight $w_H = \sum_{i=1}^{n} v_i$

- $w_L \in \mathbb{Z}^+$
  - Only certain integer combinations can sum to w_L.
  - Mathematicians call this "**integer partitioning**."
  - We care about only increasing-integer sums →

$$\begin{aligned} 6 &= 0+6 \text{ one term in sum} && w_H = 1 \\ 6 &= 1+5 \text{ two terms in sum} && w_H = 2 \\ 6 &= 2+4 \text{ two terms in sum} && \\ 6 &= 1+2+3 \text{ three terms in sum} && w_H = 3 \end{aligned}$$

Integer Partitions of interest

- Search is only up to a certain w_H (or abort-search point).
  - More than w_H probably means there is an error anyway.
  - This presumes **pre-ordered** with lowest integers most likely included in sum.
  - In other words, "don't bother checking sums that are too large" – it's an error already nothing you can do to fix.

$$w_L = \sum_{i=1}^{w_H} i = \frac{w_H \cdot (w_H + 1)}{2}$$

- Further, w_L also limits w_H .
  - This solves quadratic above using largest w_L.

$$w_H \leq \left\lfloor \frac{\sqrt{1 + 8 \cdot w_L} - 1}{2} \right\rfloor$$

- For a given w_L (max |LLR| sum):
  - Many error patterns can be omitted because they cannot add to this value.
  - Check increasing w_L up to a point that is prudent (then abandon).
- Search Increments w_L and repeats for each w_H in range.
- Search continues until codeword match or abort.

Note: This low-SNR search can precompute the guess test patterns!

Or, it's pretty easy to compute in real-time (save memory - Landslide)

Landslides and Mountains

Section 7.1.2

Further number-of-guesses reduction

- The w_H positions where the flips occur are $1 \leq a_1 \leq \cdots \leq a_{w_H} \leq n$.

- These positions map to a (also increasing order) a smaller-magnitude increasing set $\{b_i \triangleq a_i - i\}$.

$$0 \leq b_1 \leq \ldots \leq b_{w_H} \leq \underbrace{n - \frac{w_H \cdot (w_H + 1)}{2}}_{n'}$$

- This compresses the sum search set to a smaller w_L' without losing any integer-partitions of interest.

$$w_L' \triangleq w_L - \frac{w_H \cdot (w_H + 1)}{2}$$

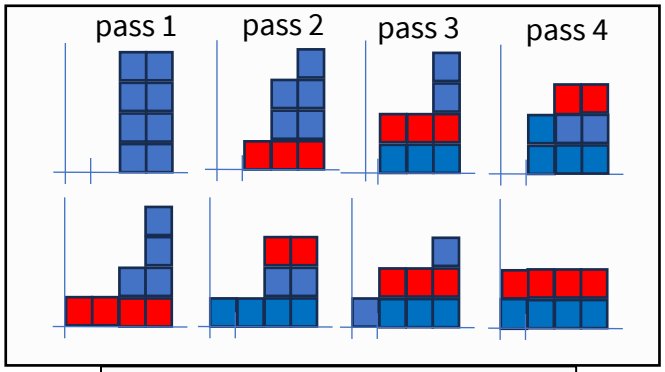
$$n' \triangleq n - w_H$$

Duffy's Landslide example

- Step 4 below is the mountain build; rest is landslide
  - $b_{1:w} = 0$ or last landslide value
  - $i' = 0$ or last landside value

Find the b_i

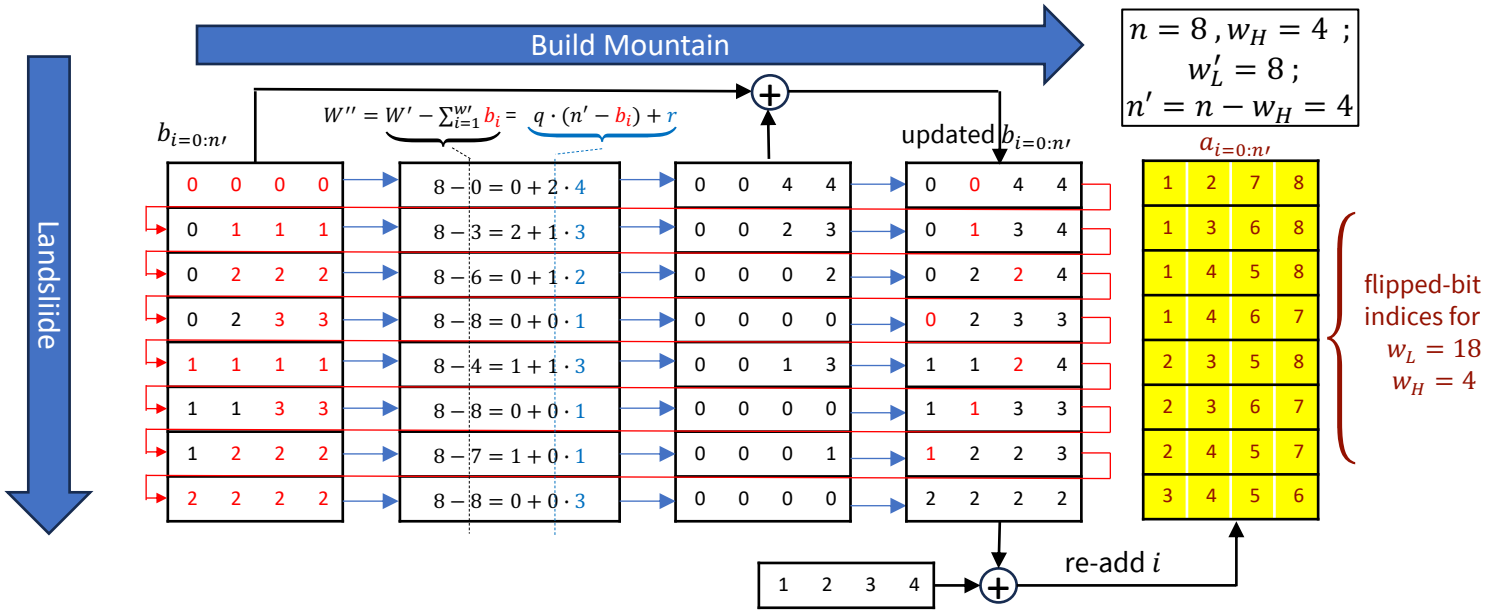
1. $b_j \leftarrow b_{i'}$ for $i' + 1 \leq j \leq w_H$. flat plateau to right
2. $w''_L = w'_L - \sum_{i=1}^{w_H} b_i$. reset residual mountain peak
3. Decompose w''_L as $w''_L = r + q \cdot (n' - b_{i'})$. Distribute residual
   r to left of these $(n' - b_{i'})$ q times on right
4. If $q \neq 0$, $b_i \leftarrow n'$ for $w_H - q + 1 \leq i \leq w_H$.
   build new full mountain
5. $b_{w_H - q} \leftarrow b_{w_H - q} + r$.
6. $i' \leftarrow^{\Delta} \arg\{\max_i \ni b_{i+1} - b_i \geq 2\}$, find new drop-by-2 point
7. $b_{i'} = b_{i'} + 1$ increment plateau level



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

- As the mountain peak slides to the left, all the increasing integer partitions of w'_L appear.
  - Increasing from left to right on integer sums until flat plateau.

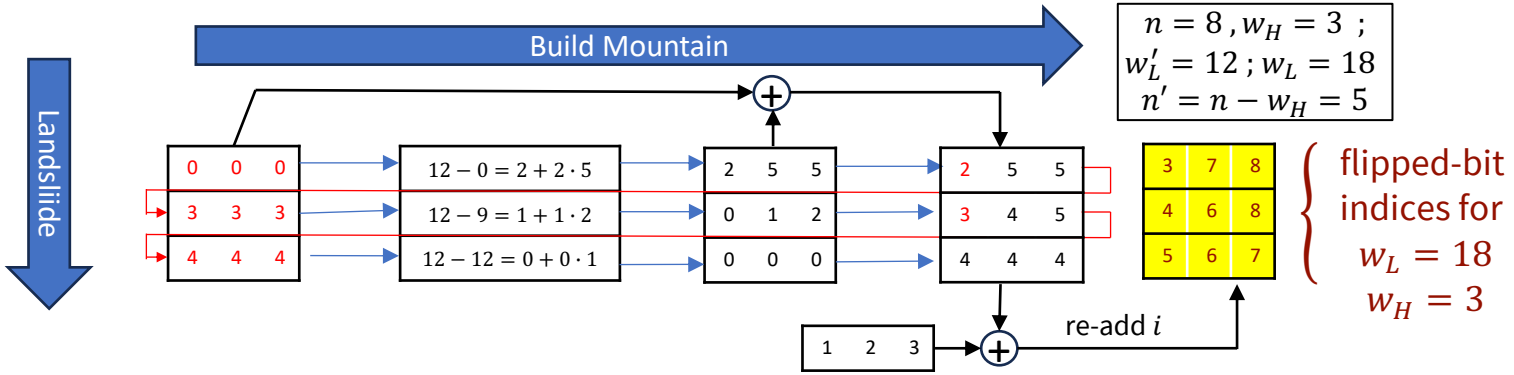
Same Example – "circuit implementation"



Type equation here.Table 4's red positions decrease by 2 (right to left), so corresponding position increases by 1 in next step, with repeats to right.
This moves the mountain top to left.

For each sequence with flipped bits, test parity.

Example with smaller Hamming



End Lecture S10A