STANFORD

Lecture 9

High-Performance Codes

February 6, 2024

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

Instructor EE379A – Winter 2024

Announcements & Agenda

- Announcements
  - PS4 due, no late. Solutions then distributed
  - Midterm on Thursday.

- Today
  - Continue L8
  - Code Performance Analysis
  - Random Interleaving
  - Iterative Decoding & Turbo Codes
  - Midterm Review

- Option & Feedback
  - Trade PS8 for any homework on grade (will give full credit on PS4.1 - 8.1e to all)
  - 11-25 hours
  - Thank you for all comments – will help future students as well.
    - I keep a running list of future corrections thanks to you all.
    - This particular assignment was on material significantly updated or new.
  - Notation (not incorrect, just a lot of it)
    - We try to avoid "one-variable-corresponding-to-multiple-things" (although ..)
    - $N = \bar{N} \cdot \tilde{N}$ is example (concatenations can confuse).
  - Problem statements
    - It helps if feedback provides a specific example of how the hmwk was not clear (which problem, what statement). We try to get that in the online feedback.
    - Sooner is better – thanks to Marcos B today
  - In class examples are simple, but homework requires more work.
  - One said HWH not helpful (of course you can ignore).
  - Time indexing
    - Vectors in communication (most recent usually on left/top)
    - Matlab has lowest index on left/top, but reverses this for G(D) octal entries
      - But not for convolution, nor convenc (and other similar commands).
- PS4 should be less
  - PS5 after midterm (probably hardest)
  - PS6-8 pretty-established problems from past (other students' comments will help you).
- No JC office hours Thursday (after exam).

Soft-Output Viterbi Algorithm
SOVA

Section 7.3.2

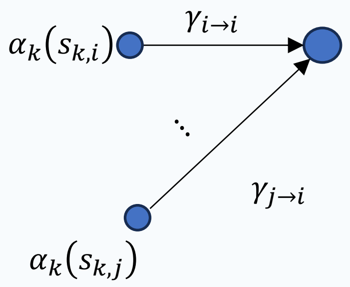
SOVA

- LOGMAX – approximates a sum-of-products by it’s maximum single term.
  - Often very true in decoding as one probability (a term) is often much larger than others (wrong decisions) PS4.3.

$$\ln(\alpha_{k+1}, s_{k+1}) \approx \max_{\text{branches into } s_{k+1}} \ln(\alpha_k, s_k, \text{branch into}) + \ln(\gamma_k, \text{branch into}) \ .$$

This is the VA in the forward direction. Similarly in the backward direction

$$\ln(\beta_k, s_k) \approx \max_{\text{branches into } s_{k+1}} \ln(\beta_{k+1}, s_k, \text{branch into}) + \ln(\gamma_k, \text{branch into}) \ .$$



$$\alpha_k(s_{k,i}) \cdot \gamma_{i\to i} + \cdots + \alpha_k(s_{k,j}) \cdot \gamma_{j\to i}$$

- The path is same as Viterbi
  - But now we have 2, one forward and one backward and try to provide better soft information about bit decisions.

$$\begin{aligned} LLR_{\boldsymbol{x}_k} \quad &= \quad \pm \Big[\max_{0 \text{ branches}} \{\ln(\alpha_k, \text{branch}) + \ln(\gamma_k, \text{branch}) + \ln(\beta_k, \text{branch})\} \\ &\qquad - \max_{1 \text{ branches}} \ln(\alpha_k, \text{branch}) + \ln(\gamma_k, \text{branch}) + \ln(\beta_k, \text{branch}) \Big] \quad , \end{aligned}$$

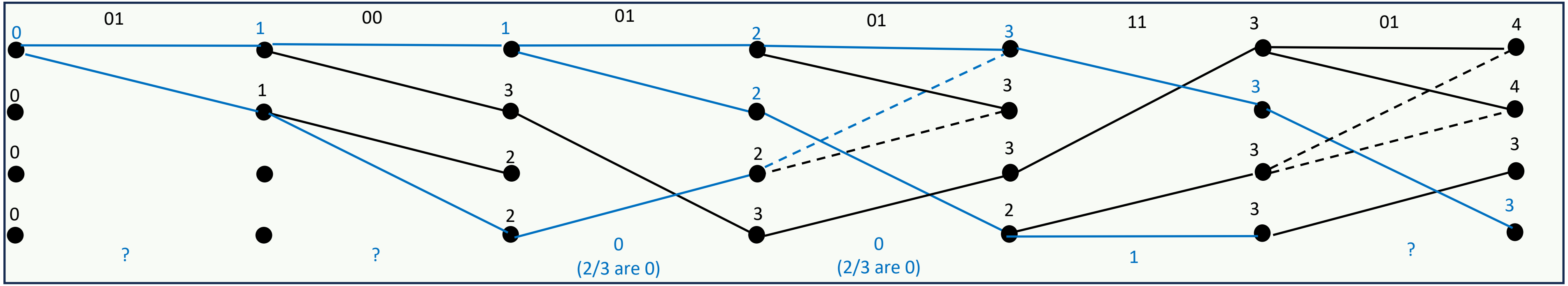
Forward SOVA Example with Ties

- It's easy without ties – just find other path with other input (0/1) with next lowest survivor metric and
  - take the difference, which magnitude (an integer for BSC) is indication of confidence (+ sign for 0 and – sign for 1)

Forward SOVA Example with ties (3-error example revisited)

k	0	1	2	3	4	5
$\{LL(0)\}$	{3}	{3}	{3,3}	{3,3}	∅	{3}
$\{LL(1)\}$	{3}	{3}	{3}	{3}	{3,3}	{3}
ΔLL (**dec**)	0(**?**)	0(**?**)	$^2/_3$ (**0**)	$^2/_3$ (**0**)	-1 (**1**)	0 (**?**)

Green color indicates the minimum-metric path is a survivor in forward direction; all LL's in units of $ln(p)$.

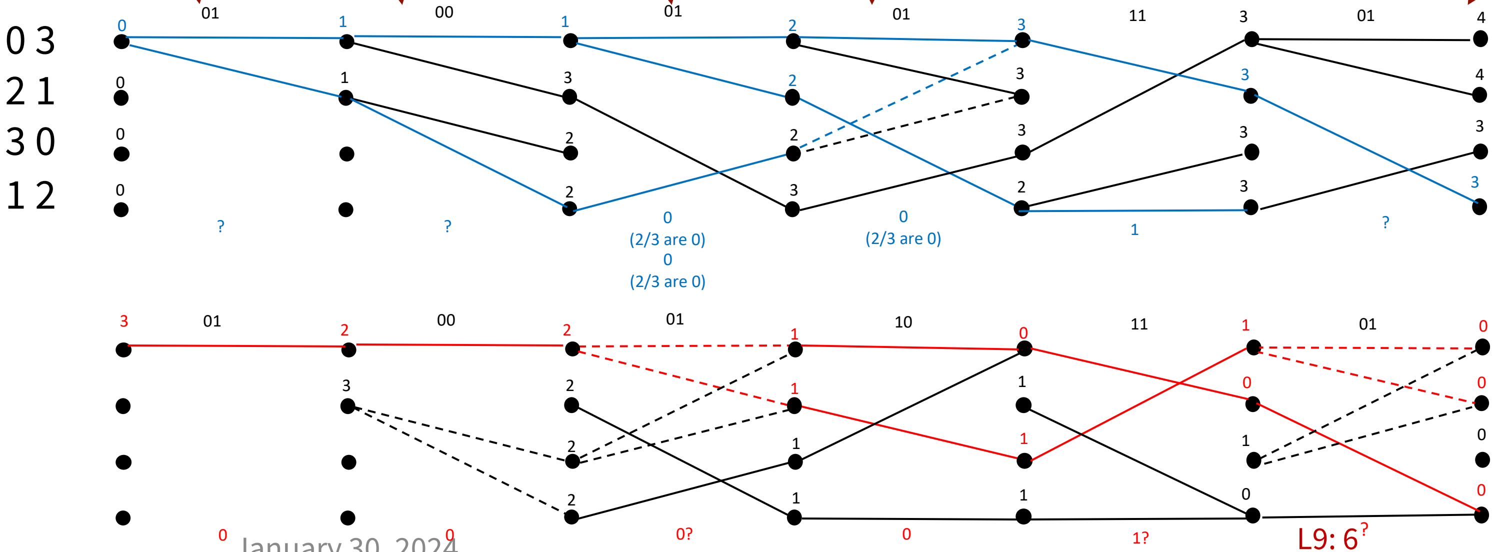


- The local resolution and majority voting appear equivalent to some of what matlab is doing (requires examination/test of source code).
  - Probably could be confirmed by someone testing various situations
  - Nonetheless, the above is viable Forward-SOVA tie resolution

Forward-Backward SOVA Example

Forward-Backward SOVA						
k	0	1	2	3	4	5
$\{LL(0)\}$	$\left\{\underbrace{3}_{0+1+2}\right\}$	$\left\{\underbrace{3}_{1+0+2}\ \underbrace{4}_{1+1+2}\right\}$	$\left\{\underbrace{3}_{1+1+1}\ \underbrace{6}_{3+2+1}\ \underbrace{4}_{2+1+1}\ \underbrace{3}_{2+0+1}\right\}$	$\left\{\underbrace{3}_{2+1+0}\ \underbrace{5}_{2+2+1}\ \underbrace{3}_{2+1+0}\ \underbrace{4}_{3+0+1}\right\}$	$\left\{\underbrace{6}_{3+2+1}\ \underbrace{5}_{3+1+1}\ \underbrace{4}_{3+0+1}\ \underbrace{4}_{2+1+1}\right\}$	$\left\{\underbrace{4}_{3+1+0}\ \underbrace{5}_{3+2+0}\ \underbrace{4}_{3+0+1}\ \underbrace{3}_{3+0+0}\right\}$
$\{LL(1)\}$	$\left\{\underbrace{4}_{0+1+3}\right\}$	$\left\{\underbrace{5}_{1+2+2}\ \underbrace{4}_{1+1+2}\right\}$	$\left\{\underbrace{3}_{1+1+1}\ \underbrace{4}_{3+0+1}\ \underbrace{4}_{2+1+1}\ \underbrace{5}_{2+2+1}\right\}$	$\left\{\underbrace{4}_{2+1+1}\ \underbrace{3}_{2+0+0}\ \underbrace{4}_{2+1+1}\ \underbrace{6}_{3+2+1}\right\}$	$\left\{\underbrace{5}_{3+2+0}\ \underbrace{4}_{3+1+0}\ \underbrace{3}_{3+0+0}\ \underbrace{3}_{2+1+0}\right\}$	$\left\{\underbrace{4}_{3+1+0}\ \underbrace{3}_{3+0+0}\ \underbrace{4}_{3+1+0}\ \underbrace{5}_{3+2+0}\right\}$
ΔLL(**dec**)	1 (**0**)	1 (**0**)	$^2/_3$ (**0**)	$^2/_3$ (**0**)	-1 (**1**)	0(**?**)

Green color indicates the minimum-metric path is a survivor in both forward and backward directions; all LL's in units of $ln(p)$



Did better than Forward

Hagenauer's LLR SOVA update

- Prob of VA sequence error

$$\boxed{\begin{array}{l} Pr_{ML}\{x_k = -1\} = Pr\{u_k = 0\} \propto e^{-LS_k^*(0)} \\ Pr_{ML}\{x_k = +1\} = Pr\{u_k = 1\} \propto e^{-LS_k^*(1)} \end{array}}$$

- Magnitude difference of two bit choices is
  - $\Delta LS_k = LS_k^*(0)$- $LS_k^*(1)$
  - $LLR_k = x_k \cdot \Delta LS_k$ (really estimate)

- Linear-code analysis: 0 in numerator:

$$P_{e,k} = \frac{e^{-LS_k^*(0)}}{e^{-LS_k^*(0)} + e^{-LS_k^*(1)}} = \frac{1}{1+e^{\Delta LS_k}}$$

- Another decoder provides

$$\widehat{LLR}_k = \ln \frac{1 - \hat{\hat{P}}_{b,k}}{\hat{\hat{P}}_{b,k}}$$

- Decoder includes soft info through:

$$\bar{P}_{b,k} \leftarrow \underbrace{\hat{\hat{P}}_{b,k}}_{\text{bit differs}} \cdot \underbrace{\frac{e^{\Delta LS_k}}{1+e^{\Delta LS_k}}}_{\text{survivor correct}} + \underbrace{(1-\hat{\hat{P}}_{b,k})}_{\text{bit same anyway}} \cdot \underbrace{\frac{1}{1+e^{\Delta LS_k}}}_{\text{survivor incorrect}}$$

- Algebra provides

$$LLR_k \leftarrow \ln\left[\frac{1+e^{\Delta LS_k + \widehat{LLR}_k}}{e^{\Delta LS_k} + e^{\widehat{LLR}_k}}\right]$$

- Ignores scaling difference between sequence and bit, so

$$\Delta LS_k \to \frac{(y_k - x_k)^2}{4 \cdot d_{free} \cdot SNR}$$

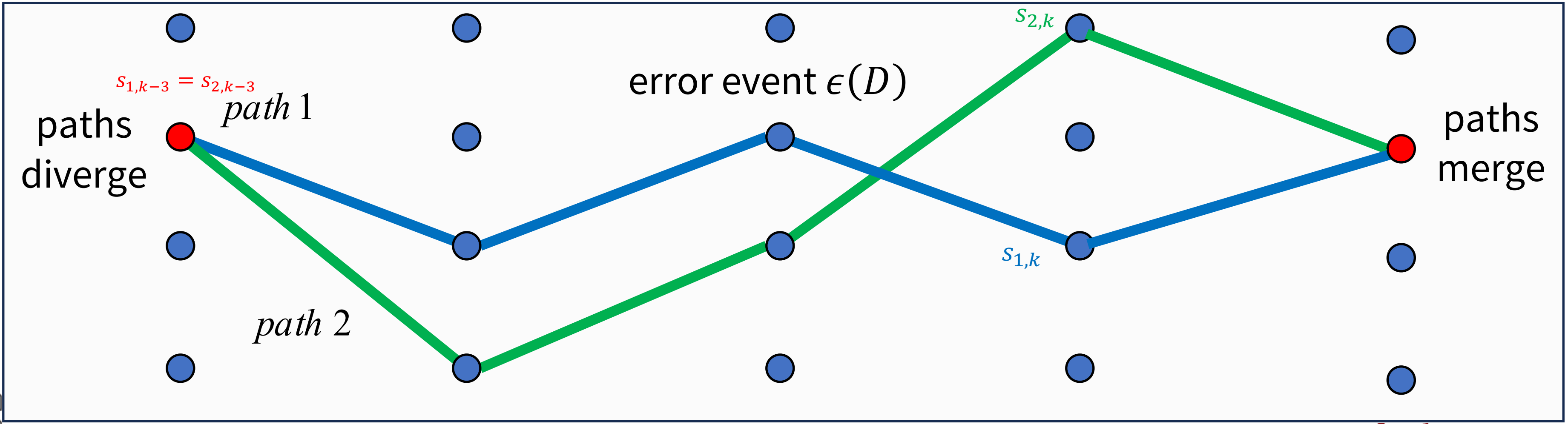
or $\Delta LS_k \to \frac{d_H(y_k, v_k)}{d_{free}}$ for BSC

Code/Decoder Performance Analysis

Section 7.2

MLSD Error Events

- MLSD is ML (maximum likelihood) – error if wrong symbol (codeword) chosen, P_e .
  - However, note MLSD's add-compare-select is basic Machine-Learning element, so ML 2 ways 😀.
- The symbol is an entire codeword, which theoretically is infinite-length for CC's.
  - An error → error event $\epsilon(D) = x(D) - \hat{x}(D)$; $\epsilon_x(D)$ with $\epsilon_y(D)$ are difference between input/output.
  - For binary codes, the subtraction is binary (mod-2 or xor).
- So, $\epsilon(D)$'s probability counts either at the time it begins (or ends, the two are equivalent) P_e .
  - All the corresponding input-bit errors are counted as occurring at that time $\bar{P}_b \geq P_e$.



Minimum distance → distance spectrum

- Union bound includes all the distances:

$$P_{e,BSC} \leq \sum_{d=d_{free}}^{\infty} N_d \cdot \left[\sqrt{4p(1-p)}\right]^d \qquad P_{e,AWGN} \leq \sum_{d=d_{free}}^{\infty} N_d \cdot Q\left(\frac{d}{2\sigma}\right)$$

- For individual bit errors, really need counting function $N(b,d)$:

$$\bar{P}_{b,BSC} \leq \frac{1}{k} \cdot \sum_{d=d_{free}}^{\infty} \sum_{i=1}^{\infty} i \cdot N(i,d) \cdot \left[\sqrt{4p(1-p)}\right]^d \qquad \bar{P}_{b,AWGN} \leq \frac{1}{k} \cdot \sum_{d=d_{free}}^{\infty} \underbrace{\sum_{i=1}^{\infty} i \cdot N(i,d)}_{N_b(d)} \cdot Q\left(\frac{d}{2\sigma}\right)$$

- Matlab finds N_d for convolutional codes (example 8-state $r = 2/3$ code) – also length (not input bit errors).

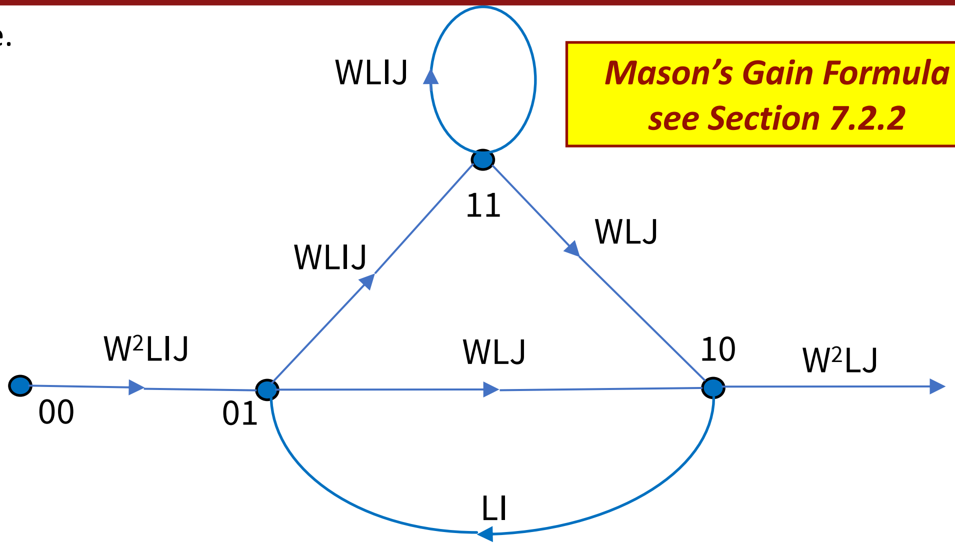
tmin=poly2trellis([3 2], [2 5 5; 3 2 1])
distspec(tmin,5) =
dfree: 4
weight: [1 11 108 417 1857] % N_b **(d)**
event: [1 5 24 71 238] % N_e (d)

d	$N_b(d)$	$N_e(d))$
4	1	1
5	**11**	5
6	**108**	**24**
7	417	71
8	1857	238

Ouch! – next-to-nearest can dominate the union-bound sum

Transfer Function Analysis (mentioned, but archaic)

- Transfer function redraws trellis as single-time state machine.
- Each branch has multivariate transfer function:
  - W^d collects distance from all zeros as exponent,
  - L^l collects length as exponent (each branch is L),
  - I^i collects input errors w.r.t. all zeros as exponent,
  - J^j collects number of subsymbol differences.



Mason's Gain Formula see Section 7.2.2

$$T(W,L,I,J) = \frac{W^5 \cdot L^3 \cdot I \cdot J^3}{1 - WLIJ \cdot (L+1)}$$

$$= W^5 \cdot L^3 \cdot I \cdot J^3 \cdot \left[1 + WLIJ \cdot (L+1) + \left(WLIJ \cdot (L+1)\right)^2 + \cdots\right]$$

- Only 1 error event has $d_{free} = 5$:
  - length is 3, with
  - 1 input bit error (Nb = **1**), &
  - 1 error event (Ne = **1**).
- **2** error events have $d = 6$:
  - lengths are 5 and 6,
  - both have 2 input bit errors (so Nb (d=6) = 4), &
  - Ne (d=6) = **2.**

```
>> t=poly2trellis(3, [7 5])
   numInputSymbols: 2
   numOutputSymbols: 4
         numStates: 4
        nextStates: [4×2 double]
           outputs: [4×2 double]
>> distspec(t,4)
   dfree: 5
   weight: [1 4 12 32]
   event: [1 2 4 8]
```

Random Interleaving

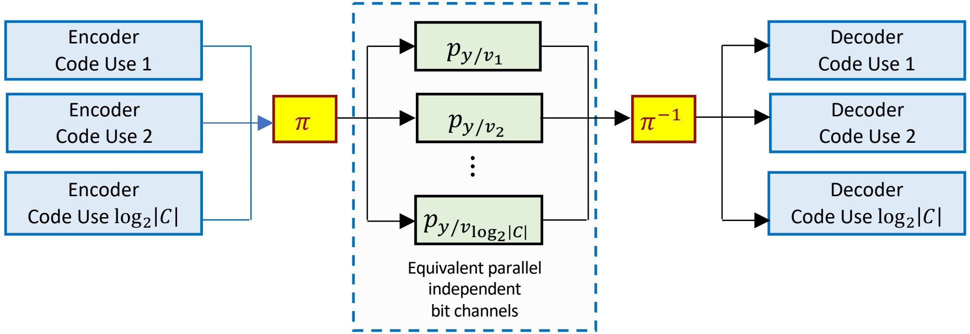
Section 8.3.1

$(\cdot)_M$ means the quantity in brackets modulo M

the part left over after subtracting the largest contained integer multiple of M.

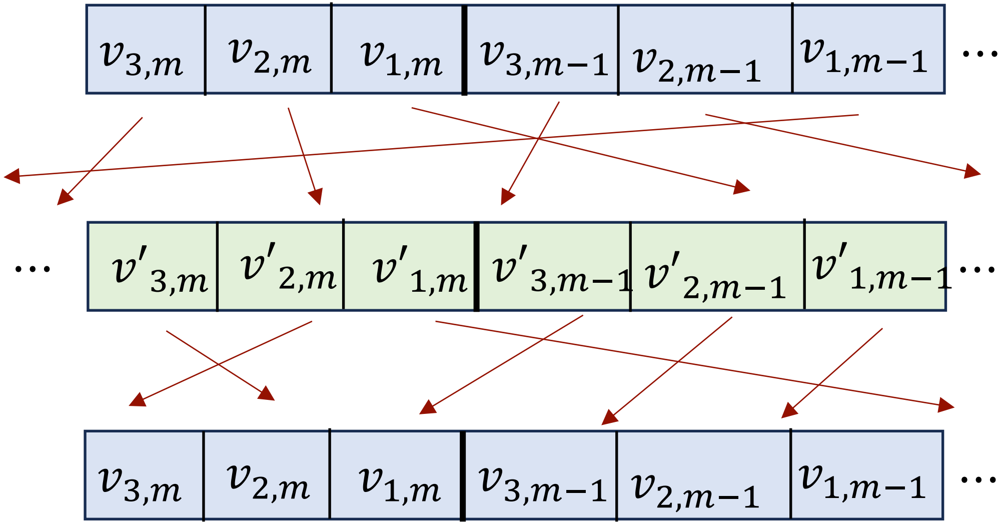
Binary Codewords & Sequences

- Recall BICM.
  - Effectively was simple/deterministic interleave (PS3.5)
- Adjacent bits are separated
  - $\boldsymbol{\pi}$ or $\pi(k)$ is periodic with period L.
  - π is also an "ordering."



Uniform Random Interleaver π

- $\binom{L}{l}$ possibilities in L encoder-output positions:
  - where l channel-bit errors could have occurred.
  - True random interleave means they're all equally likely.
- Adjacent bits are separated
  - Ideally by more than codeword length, big L>> N .
  - Or > survivor length with Viterbi.
- Receiver de-interleaves/restores original order.
  - Now coming from independent channel uses.
- Real system, L is delay, so not too big.



Various implementations of random interleavers

- **Berrou Glavieux:** $L = K \cdot J = 2^i \cdot 2^j$

$r_0 = (k)_J,\ c_0 = (k - r_0)/J$ @ *time k*

$$\begin{aligned} r(k) &= (p_{m+1} \cdot (c_0 + 1) - 1)_K \\ c(k) &= ((K/2 + 1) \cdot (r_0 + c_0))_J \end{aligned}$$

m	1	2	3	4	5	6	7	8
p_m	17	37	19	29	41	23	13	7

$$\pi(k) = c(k) + J \cdot r(k)$$

- **JPL Block Interleaver:** $K \in \mathbb{Z}^+$ and even $J \in \mathbb{Z}^+$, such that $L = K \cdot J$

$$\begin{aligned} r(k) &= (19 \cdot r_0)_{\frac{K}{2}} \\ c(k) &= (p_{m+1} \cdot c_0 + 21 \cdot (k)_2)_J \end{aligned}$$

on is

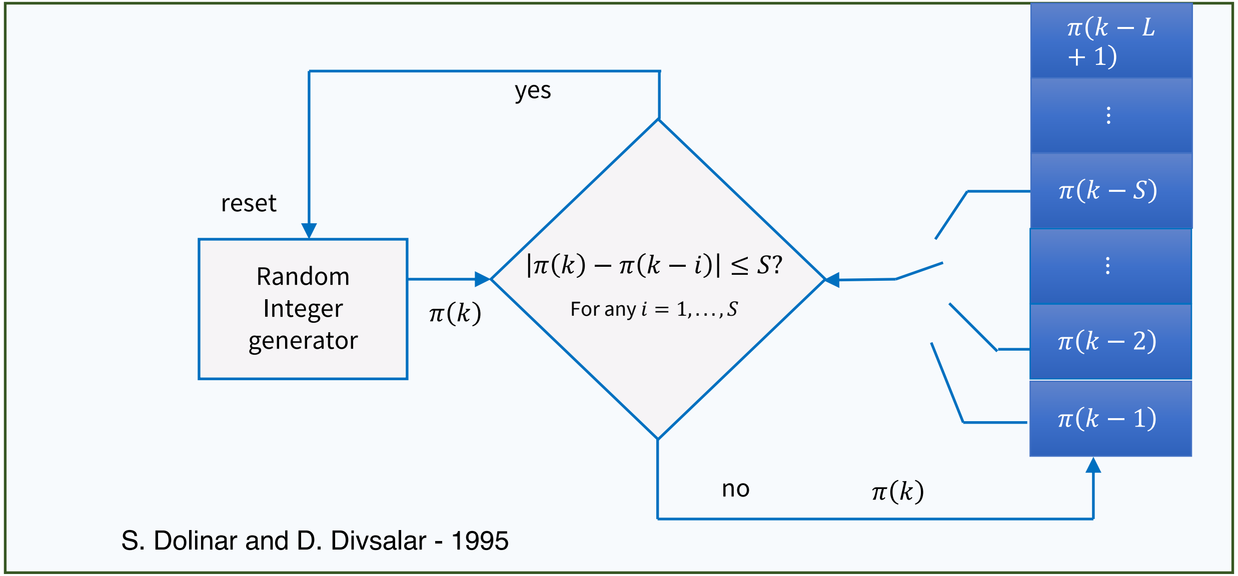
m	1	2	3	4	5	6	7	8
p_m	31	37	43	47	53	59	61	67

$$\pi(k) = 2 \cdot r(k) \cdot K \cdot c(k) - (k)_2 + 1$$

$r_0 = \left(\frac{i-m}{2} - c_0\right)_J$, $c_0 = \left(\frac{i-m}{2}\right)_J$, and $m = (r_0)_8$

- **Pseudorandom sequence:** ν bits – visits every ν –bit sequence exactly once in $L = 2^\nu - 1$ **period**.
  - These are generated with rate $r = 1,\ 2^\nu$ –state $G(D)$=1/$P(D)$ encoder.
  - Table 8.6 lists the $P(D)$ choices for different ν choices or use matlab's prbs.m command.

S-random interleaver – adapts to situation



- $S \leq \sqrt{\frac{L}{2}}$, run loop with largest S that provides enough values for $\pi(k)$.
- There is no best universal choice of random interleaver – it is related to code choice(s).
- Ileaveout(1:L) = Ileavein(pi(1:L)) ; ileavein = ileaveout(piinv(1:L)) – for design's pi choice .

Matlab random interleaving

- Matlab has its own function where a state can be selected $0 \le STATE < 2^{31}$; $STATE \in \mathbb{Z}$

  INTRLVED = randintrlv(DATA, STATE)

  DATA=randintrlv(STATE)

- STATE needs to be the same in both calls. STATE evolves for successive DATA packets:
  - Use randi (random integer) or randStream commands for STATE.
- **This appears S-random** for length of data and seems to approximate the $\binom{L}{l}$ possible patterns for $l \ll L$,
  - with roughly equal probability (L=length(DATA)).

```
>> data=randi(2,[1,8])-1 =
   1   0   1   1   1   1   1   0
>> ileaveout=randintrlv(data,1)=
   1   0   1   0   1   1   1   1
>> randdeintrlv(ileaveout,1) =
   1   0   1   1   1   1   1   0
```

```
>> data=randi(2,[1,10])-1 =
   0   1   1   0   1   0   0   1   1   1

>> ileaveout=randintrlv(data,1) =
   1   1   0   1   0   1   0   1   0   1

>> randdeintrlv(ileaveout,1) =
   0   1   1   0   1   0   0   1   1   1
```

- **PRBS Version:**

```
>> prbs(3,7) =    0   1   0   1   1   1   0
>> prbs(3,14) =
0
>> sum(prbs(3,14)) =    8
>> sum(prbs(3,14) == 0) =    6
```

5 7 4

0 1 0 1 1 1 0 0 1 0 1 1 1

2 3 6 1

```
>> randperm(7) =   6   3   7   5   1   2   4
```

Nothing Perfectly Random on interleaving.

- **randperm**(L) - used in coming examples.

Iterative Decoding & Turbo Codes

Section 8.3.2

Intrinsic and Extrinsic Soft Information

$$p_{\boldsymbol{x}_k/\boldsymbol{Y}_{0:K-1}} \propto p_{\boldsymbol{x}_k,\boldsymbol{Y}_{0:K-1}} = \underbrace{\underbrace{p_{\boldsymbol{x}_k}}_{\text{à priori}} \cdot \underbrace{p_{\boldsymbol{y}_k/\boldsymbol{x}_k}}_{\text{channel}}}_{\text{intrinsic}} \cdot \underbrace{p_{\boldsymbol{Y}_{0:k-1},\boldsymbol{Y}_{k+1:K-1}/\boldsymbol{x}_k,\boldsymbol{y}_k}}_{\text{extrinsic}} \ .$$

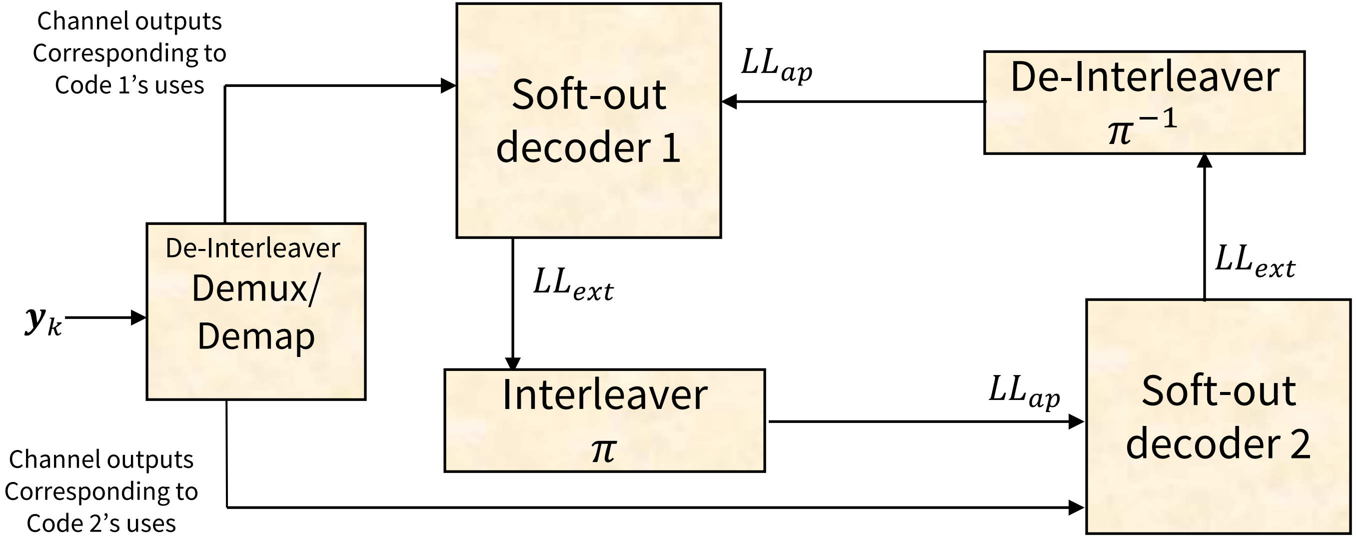
γ_k $\qquad$ α_k , β_k

- **Intrinsic soft information** (at time k):
  - includes à priori information (which initially is uniform, but often replaced by another decoder's update).
  - includes the channel output (examples squared distance to closest constellation point AWGN, ln(p(1-p)) on BSC).
- **Extrinsic soft information** (at times $\neq k$)**:**
  - Includes everything else from code (and sometimes channel/constellation-mapping) constraints.
  - For instance, the MAP decoder's α_k , β_k that accumulate information for all the other subsymbols in codeword.
- The **next** à priori becomes the **last** extrinsic in successive decoding iterations.

Iterative Decoding with LL's

- Decompose LLs (or often LLRs for bit decoding)

$$
\begin{aligned}
LL_{\boldsymbol{x}_k,\boldsymbol{Y}_{0:K-1}} &= LL_{\boldsymbol{x}_k} + LL_{\boldsymbol{y}_k/\boldsymbol{x}_k} + LL_{\boldsymbol{Y}_{0:k-1},\boldsymbol{Y}_{k+1:K-1}/\boldsymbol{x}_k,\boldsymbol{y}_k} \\
&= \underbrace{LL_{\text{à priori}} + LL_{\text{channel}}}_{\text{bias accumulation risk}} + LL_{\text{extrinsic}} \quad .
\end{aligned}
$$



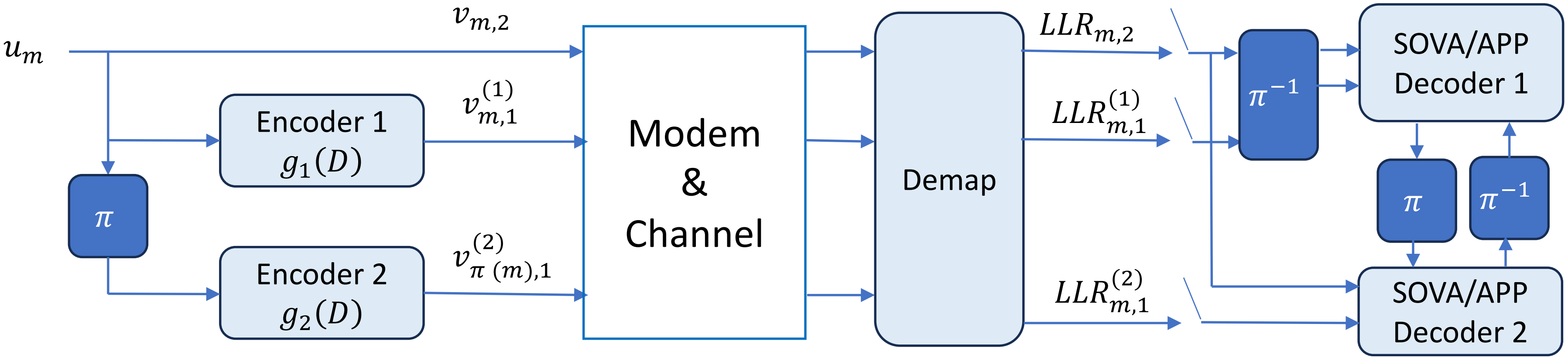
Simple at high level, details can be complex

e.g.
Are the LLR's for the encoder input or output bits or both?
(it depends)

- Converged if two have same decision (or exceed interation max)

Parallel Code Concatenation

- **Parallel Concatenation** – usually occurs with $r = 1/n$ codes and systematic encoders.
  - Often $g_1(D)$= $g_2(D)$ and is called a "**Turbo Code**."



- Overall rate is r=1/3, but each is r=1/2 in the example above.
  - If nonsystematic, the efficiency of not repeating the same bit is lost.

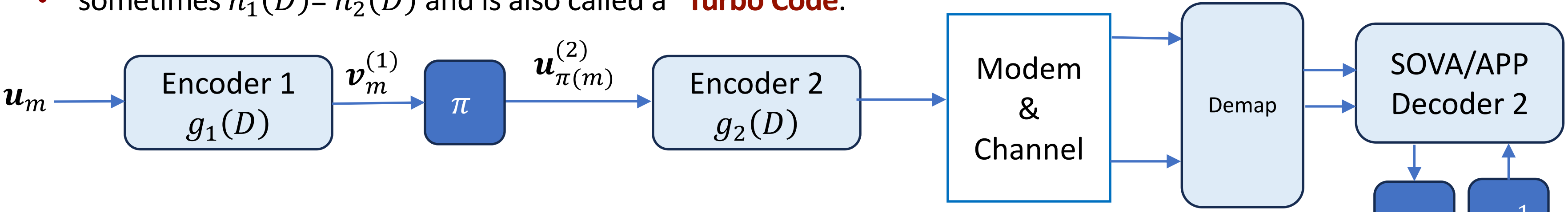
$$\frac{1}{r} = \frac{1}{r_1} + \frac{1}{r_2} - \begin{cases} 1 & systematic \\ 0 & nonsystem \end{cases}$$

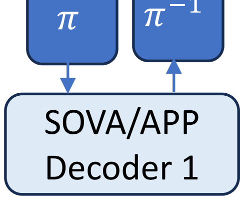
See footnote in Section 8.3 for more on this formula.

- Soft extrinsic information between decoders is for input bits.
- *Initial* soft intrinsic information is for channel/encoder output bits.

Serial Code Concatenation

- Serial Concatenation – usually occurs with $r = \frac{k}{k+1}$ codes and a systematic encoder.
  - sometimes $h_1(D)$= $h_2(D)$ and is also called a "**Turbo Code**."





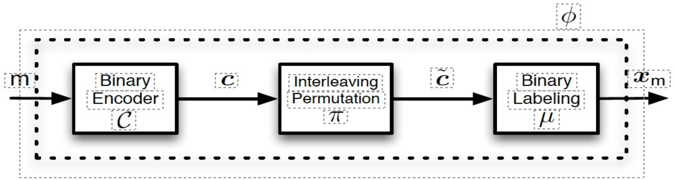
- Overall rate is $r = r_1 \cdot r_2$.

- Soft extrinsic information between decoders has code 1's out bits and code 2's in bits.

- *Initial* soft intrinsic information is for code 2's channel/encoder output bits.

BICM is simple Example,
but can be 2 more complex codes.

So are "turbo equalizers"
and "multiuser soft cancellers."



Albert Guillén i Fàbregas, Alfonso Martinez and Giuseppe Caire (2008), "**Bit-Interleaved Coded Modulation**", Foundations and Trends® in Communications and Information Theory: Vol. 5: No. 1–2, pp 1-153.
http://dx.doi.org/10.1561/0100000019

Puncturing (parallel case)

- Parity bits here may be for different codes with same input bit(s).

- This restores code rate $r = \frac{1}{3}$ to $\frac{1}{2}$

u_k	$p_{1,k}$	$\cancel{p_{2,k}}$	u_{k+1}	$\cancel{p_{1,k+1}}$	$p_{2,k+1}$

- increases $r = \frac{1}{3}$ to $\frac{2}{3}$

u_k	$p_{1,k}$	$\cancel{p_{2,k}}$	u_{k+1}	$\cancel{p_{1,k}}$	$\cancel{p_{2,k}}$	u_{k+2}	$\cancel{p_{1,k+2}}$	$p_{2,k+2}$	u_{k+3}	$\cancel{p_{1,k+3}}$	$\cancel{p_{2,k+3}}$

- Serial case similar – try to distribute deleted bit positions equally throughout

Turbo Codes

- Individual codes are convolutional codes (parallel and serial cases).

- Example analysis for well known 4-state $r = \frac{1}{2}$ code:

$$\boxed{G_1(D) = \left[1 \quad \frac{1+D+D^2}{1+D^2}\right]}$$

- An input error event $1 + D^2$ for 1st-code corresponds to output error event $[1 + D^2 \quad 1 + D + D^2]$,
  - which is the $d_{free} = 5$ error event (or closest codeword to all zeros).
  - So there are **2 input-bit** errors if an **output-bits error event** causes decoder to pick wrong codeword.
- But this has to happen for 2nd-code also (same code, just a 2nd parity bit);
  - so, $d_{free} = 8$ for the concatenated code.
    - The new gain is γ =8/3 instead of 5/2, which is roughly only .3 dB improvement; HOWEVER

$$Prob\ both\ err = \binom{L}{2}^{-1} = \frac{2}{L \cdot (L-1)}$$

$$\begin{aligned} \bar{P}_b(d_{free}) &\approx \frac{2}{L-1} \cdot b \cdot \overline{N}_b(d_{free}) \cdot Q(\sqrt{d_{free} \cdot \mathrm{SNR}}) \\ &\approx \frac{4}{L-1} \cdot 1 \cdot Q(\sqrt{8 \cdot \mathrm{SNR}}) \quad . \end{aligned}$$

- So N_b reduces by $2/(L-1)$; if $L = 2000$, this is ~ 2-3 dB improvement, $\sim 0.2 \cdot \log_2 L$ in range $10^{-4} - 10^{-7}$.
  - This basic effect occurs also with more powerful convolutional codes, but still about 1-1.5 dB short of best (capacity).

Matlab’s comm.TurboEncoder/Decoder

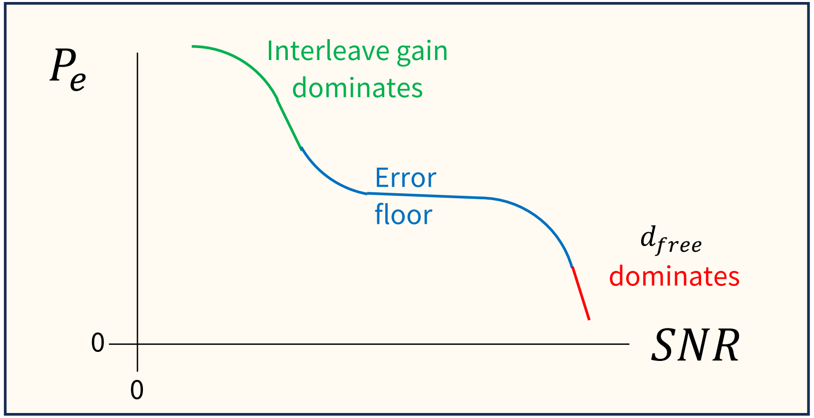
```
>> tfeed=poly2trellis(3,[7 5],7) =
        numInputSymbols: 2
        numOutputSymbols: 4
        numStates: 4
        nextStates: [4 × 2 double]
        outputs: [4 × 2 double]

>> ileaveorder=randperm(100);

>> turbo = comm.TurboEncoder('TrellisStructure', tfeed, 'InterleaverIndices',
ileaveorder) =
        TrellisStructure: [1 × 1 struct]
        InterleaverIndicesSource: 'Property'
        InterleaverIndices: [99 32 40 22 34 … ]
        OutputIndicesSource: 'Auto'   % this deletes the repeated input bit.

>> X=prbs(7,100);
>> Y=turbo(X ')' =
001 000 001 …
>> size(Y) =   1  308  % This equals (1/r) x L plus 2 x nu x n
```

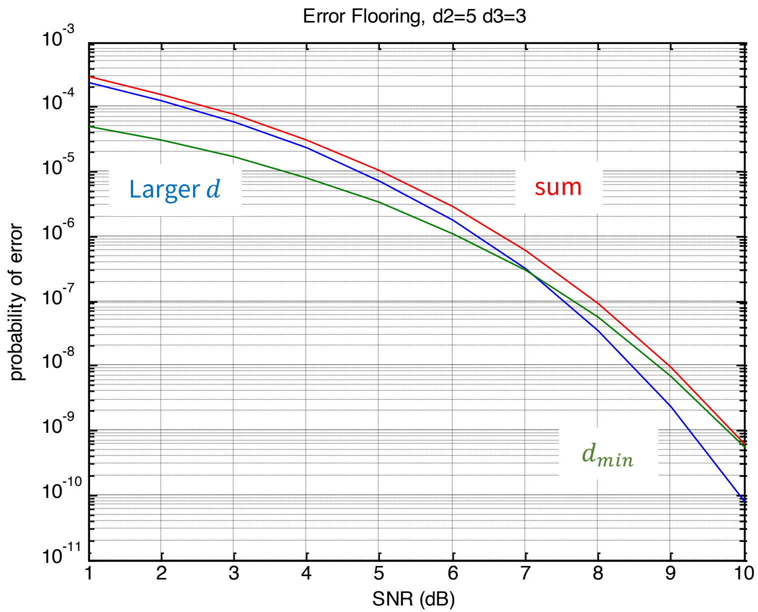
```
>> turbodec=comm.TurboDecoder(tfeed,ileaveorder)
        TrellisStructure: [1 × 1 struct]
   InterleaverIndicesSource: 'Property'
        InterleaverIndices: [99 32 40 22 34 ]
        InputIndicesSource: 'Auto'
              Algorithm: 'True APP'
           NumIterations: 6
>> decmsg=turbodec(2*Y-1); % takes real numbers with 0→ -1,  1→ +1
>> (decmsg-X')'  =
   000  000  000  000  000  000  000  000  000  000  000  000 000  000 000
   000  000  000  000  000  000  000  000  000  000  000  000 000  000 000
   000  000  000  0
>> >> biterr(decmsg,X') =    0

>> error = [ 1 zeros(1,49) 1 zeros(1,49) 1 zeros(1,99) 1 zeros(1,45) 1 zeros(1,61)];
>> decmsg=turbodec(2*(+xor(Y,error))-1);
>> (decmsg-X')’
   000  000  000  000  000  000  000  000  000  000  000  000 000  000 000
   000  000  000  000  000  000  000  000  000  000  000  000 000  000 000
   000  000  000  0
>> biterr(decmsg,X') =    0
```

- Decode tool default is MAP/APP Decode.
  - Options can set max (max) and maxlog (max*).
  - Options can increase number of bits also (NumScalingBits) up to 8 (default is 3-bit arithmetic).
  - Soft information on p or σ^2 is consistent within and need not be supplied (only need this for MAP when sending outside this loop).

Error Flooring

- Interleaver gain additional gain is $\gamma_{parallel} \cong \log_{10}\left(\frac{L-1}{2}\right)$ dB
  - over the range, $\gamma_{parallel}$ < 3 dB .
- $\gamma_{parallel}$ dominates over operational range where $d > d_{min}$ next-to-nearest counts contribute significantly.
- At large SNR, the d_{min} term in Q-function argument(s) eventually dominates.

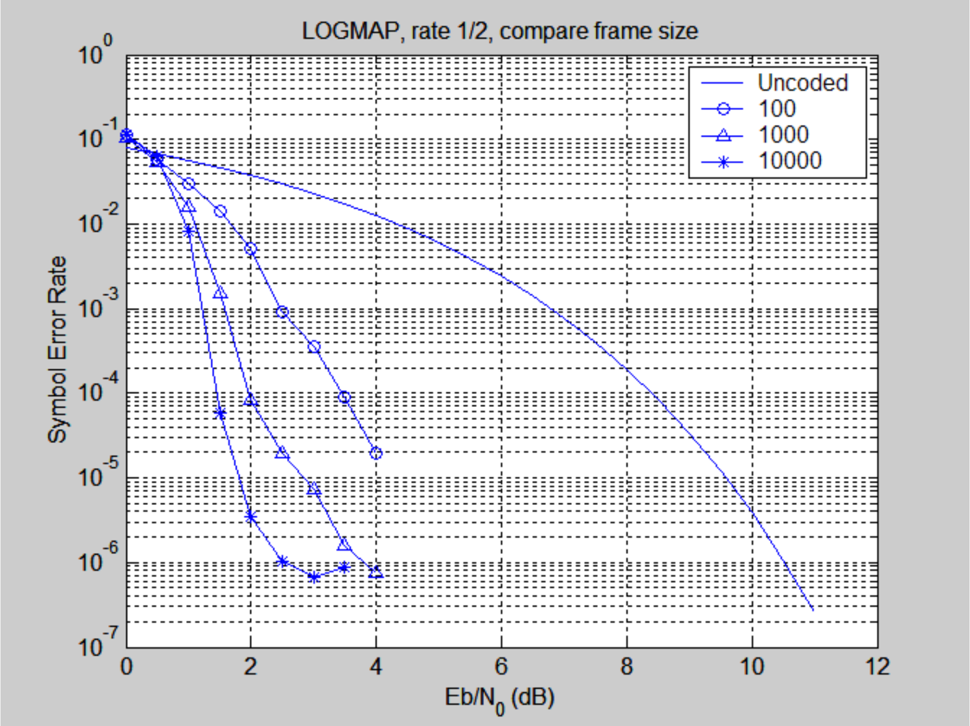




Limits coding gain or equivalently errors may not go to zero when SNR is better than anticipated in design.

Section 8.3.2.3

The example 4-state code with puncturing



We're almost to $\mathcal{C}$!

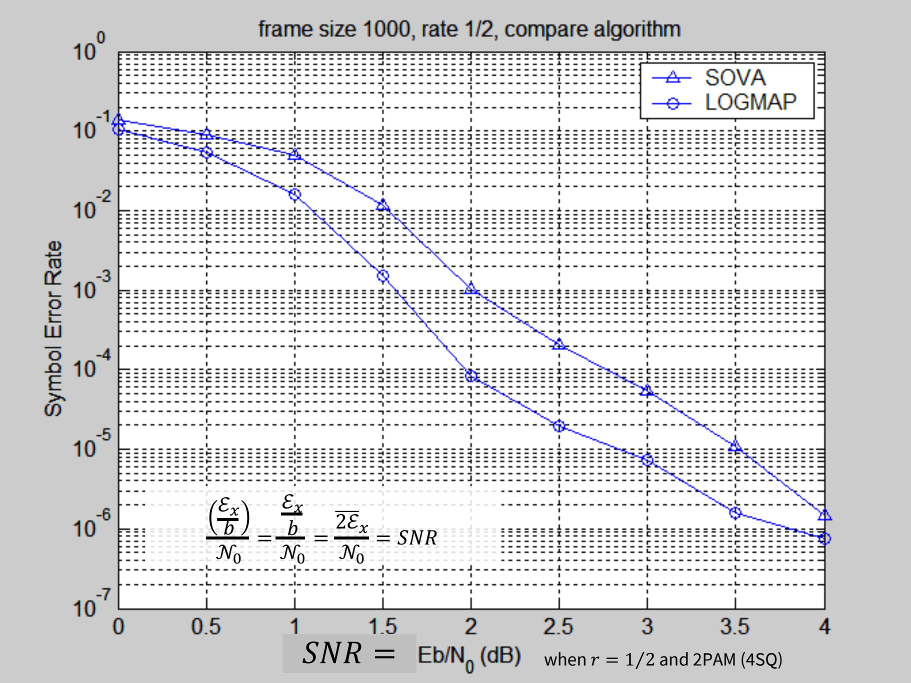
Certain high-rate (see L11, EE387) Reed Solomon byte-wise block codes can, with small overhead, take $\bar{P}_b = 10^{-6}$ close to zero

The drop in $\bar{P}_b = 10^{-6}$ becomes very steep, so coding gain becomes less meaningful – high sensitivity to small SNR changes (there is another interleaver outside).

But that gets us close enough to capacity

- The gain at $\bar{P}_b = 10^{-6}$ is about 8.5 dB ; however $\mathcal{C} = 0.5$ @ $SNR = 0$ dB (so still about 2 dB short)
  - But roughly 3 dB better than best 16-state $r = {}^{1}/_{2}$ code!

SOVA / Logmax and error flooring



- SOVA w.r.t. LOGMAP loses about 0.5 dB and both show earlier error flooring.

Good constituent Turbo Codes (Divsilar, JPL)

- Section 8.3 lists many.

2^ν	$g_0(D)$	$g_1(D)$	d_2	d_3	d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
4	7	5	5	6	5	8	10	8
8	17	13	6	7	6	10	11	10
16	23	35	7	7	7	12	11	11

Table 8.10: Rate 1/2 constituent (mother) parallel convolutional codes

Fit to design, but probably at best another 1 dB over the earlier example.

Some may be better for BICM outside of Turbo.

2^ν	$g_0(D)$	$g_1(D)$	$g_2(D)$	d_2	d_3	d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
2	3	2	1	4	∞	4	6	∞	6
4	7	5	3	8	7	7	14	11	11
8	13	17	15	14	10	10	26	17	17
16	23	33	37	22	12	12	42	21	21

Table 8.11: Rate 1/3 constituent (mother) parallel convolutional codes.

2^ν	$h_0(D)$	$h_1(D)$	$h_2(D)$	d_2	d_3	d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
4	7	3	5	4	3	3	6	3	3
8	13	15	17	5	4	4	8	5	5
16	23	35	27	8	5	5	14	7	7
16	45	43	61	12	6	6	22	9	9

Table 8.13: Rate 2/3 constituent (mother) parallel convolutional codes.

Good Punctured Codes & Patterns (DLM) for Turbo

- The puncturing (as well as interleaver, and of course the 2 together) is at least as important as the code.

2^ν	$\left[1\ \frac{g_1}{g_0}\right]$	d	$N_{e,d}$	$N_{b,d}$	d_2	d_3
4 (d_2)	$[1\ \frac{5}{7}]$	5	1	3	6	5
		6	2	6		
		7	4	14		
		8	8	32		
		9	16	72		
4 (SNR)	$[1\ \frac{7}{5}]$	5	1	2	5	∞
		6	2	6		
		7	4	14		
		8	8	32		
		9	16	72		
8 (d_2)	$[1\ \frac{15}{13}]$	6	2	6	8	6
		8	10	40		
		10	49	245		
		12	241	1446		
		14	1185	8295		
8 (d_2)	$[1\ \frac{17}{13}]$	6	1	4	8	7
		7	3	9		
		8	5	20		
		9	11	51		
		10	25	124		
8 (SNR)	$[1\ \frac{15}{17}]$	6	1	2	6	∞
		7	3	12		
		8	5	20		
		9	11	48		
		10	25	126		
16 (d_2)	$[1\ \frac{33}{31}]$	7	2	8	12	7
		8	4	16		
		9	6	26		
		10	15	76		
		11	37	201		
16	$[1\ \frac{21}{37}]$	6	1	2	6	∞
		7	1	5		
		8	3	10		
		9	5	25		
		10	12	56		

2^ν	$\left[1\ \frac{g_1}{g_0}\right]$	d	$N_{e,d}$	$N_{b,d}$	d_2	d_3
16 (d_2)	$[1\ \frac{27}{31}]$	7	2	8	12	7
		8	3	12		
		9	4	16		
		10	16	84		
		11	37	213		
16 (d_2)	$[1\ \frac{37}{23}]$	6	1	4	12	8
		8	6	23		
		10	34	171		
		12	174	1055		
		14	930	6570		
16 (d_2)	$[1\ \frac{33}{23}]$	7	2	8	12	7
		8	4	16		
		9	6	26		
		10	15	76		
		11	37	201		
16 (d_2)	$[1\ \frac{35}{23}]$	7	2	8	12	7
		8	3	12		
		9	4	16		
		10	16	84		
		11	37	213		
16 (SNR)	$[1\ \frac{23}{35}]$	7	2	6	7	∞
		8	3	12		
		9	4	20		
		10	16	76		
		11	137	194		
32 (d_3)	$[1\ \frac{71}{53}]$	8	3	12	12	∞
		10	16	84		
		12	68	406		
		14	860	6516		
		16	3812	30620		
32 (SNR d_2)	$[1\ \frac{67}{51}]$	8	2	7	20	8
		10	20	110		
		12	68	406		
		14	469	3364		
		16	2560	20864		

Table 8.15: Best 4/8/16/32-state $r = 1/2$ constituent (mother) convolutional codes with puncturing. $N_{e,d} = N_{d-d_{free}}$ and $N_{b,d} = \sum_b N(b,d)$ in this table.

$n-1$	4 states	8 states	16 states	32 states
2 $^2/_3$	$[1\ \frac{5}{7}]$ 13 (3,1,3) $d_2=4$, $d_3=3$	$[1\ \frac{15}{13}]$ 13 (4,3,10) $d_2=5$, $d_3=4$	$[1\ \frac{37}{23}]$ 13 (4,2,6) $d_2=7$, $d_3=4$	$[1\ \frac{67}{51}]$ 13 (5,2,7) $d_2=9$, $d_3=5$
3 $^3/_4$	$[1\ \frac{5}{7}]$ 56 (3,4,10) $d_2=3$, $d_3=3$	$[1\ \frac{15}{13}]$ 53 (3,2,5) $d_2=3$, $d_3=3$	$[1\ \frac{37}{23}]$ 53 (3,1,3) $d_2=4$, $d_3=3$	$[1\ \frac{67}{51}]$ 53 (4,2,7) $d_2=7$, $d_3=4$
4 $^4/_5$	$[1\ \frac{5}{7}]$ 253 (2,1,2) $d_2=2$, $d_3=3$	$[1\ \frac{15}{13}]$ 253 (3,9,24) $d_2=3$, $d_3=3$	$[1\ \frac{37}{23}]$ 253 (3,3,9) $d_2=4$, $d_3=3$	$[1\ \frac{67}{51}]$ 253 (3,1,3) $d_2=5$, $d_3=3$
5 $^5/_6$	$[1\ \frac{5}{7}]$ 1253 (2,2,4) $d_2=2$, $d_3=3$	$[1\ \frac{17}{13}]$ 1253 (3,15,40) $d_2=3$, $d_3=3$	$[1\ \frac{27}{31}]$ 1272 (3,2,6) $d_2=4$, $d_3=3$	$[1\ \frac{71}{53}]$ 1272 (4,108,406) $d_2=4$, $d_3=\infty$
6 $^6/_7$	$[1\ \frac{5}{7}]$ 5352 (2,22,44) $d_2=2$, $d_3=3$	$[1\ \frac{17}{13}]$ 5253 (2,1,2) $d_2=2$, $d_3=3$	$[1\ \frac{27}{31}]$ 5253 (3,12,33) $d_2=3$, $d_3=3$	$[1\ \frac{71}{53}]$ 5253 (3,3,6) $d_2=3$, $d_3=\infty$
7 $^7/_8$	$[1\ \frac{5}{7}]$ 25253 (2,7,14) $d_2=2$, $d_3=3$	$[1\ \frac{15}{17}]$ 25253 (2,7,14) $d_2=2$, $d_3=\infty$	$[1\ \frac{33}{23}]$ 25253 (2,1,2) $d_2=2$, $d_3=3$	$[1\ \frac{67}{51}]$ 25253 (2,1,2) $d_2=3$, $d_3=3$
8 $^8/_9$	$[1\ \frac{5}{7}]$ 125253 (2,9,18) $d_2=2$, $d_3=3$	$[1\ \frac{15}{13}]$ 125253 (2,4,8) $d_2=2$, $d_3=3$	$[1\ \frac{37}{23}]$ 125253 (2,1,2) $d_2=2$, $d_3=3$	$[1\ \frac{67}{51}]$ 125253 (3,17,49) $d_2=3$, $d_3=3$

Table 8.16: Best puncturing patterns for given high-rate parallel turbo codes. The triplets listed are $(d_i, N_{d-d_{free}}, \sum_b N(b,d))$.

0 meaning puncture that parity bit. For instance 5352 means 101 011 101 010 so and corresponds to (letting i_k be an information bit and p_k be the corresponding parity bit)

$$(i_1, p_1, i_2, p_2, i_3, p_3, i_4, p_4, i_5, p_5, i_6, p_6) \rightarrow (i_1, i_2, i_3, p_3, i_4, i_5, i_6) \quad . \tag{8.66}$$

Serial Concatenation

- It may be easier to concatenate with an existing system serially.
  - Use when necessary (essentially pass-through when SNR is above minimum necessary – systematic, no decode).
- Analysis is more complex, see Section 8.3.

$$\bar{P}_b \approx L \left(\frac{L}{\left\lceil \frac{d_{free}^{out}}{2} \right\rceil} \right)^{-1} \cdot \frac{\bar{N}_b(d_{free}^{in}) \cdot \bar{N}_b(d_{free}^{out})}{b} \cdot Q\left(\sqrt{\left\{ \left\lceil \frac{d_{free}^{out} - 3}{2} \right\rceil \cdot d_2^{in} + d_w^{in} \right\} \cdot \text{SNR}} \right) \quad , \qquad (8.64)$$

This analysis does not require the outer encoder to be systematic, nor even use feedback. The interleaving gain thus is

$$\gamma_{serial} = \log_{10} \left(\frac{L!}{\left\lceil \frac{d_{free}^{out}}{2} \right\rceil !} \right) \text{ dB} \qquad (8.65)$$

Which is better, parallel or serial?
Really depends on situation, exact SNR.
Turbo codes have largely yielded to LDPC codes in recent years (next lecture).

Midterm Review

STANFORD

End Lecture 9