

Turbo and LDPC Codes: Implementation, Simulation, and Standardization

June 7, 2006

Matthew Valenti
Rohit Iyer Seshadri
West Virginia University
Morgantown, WV 26506-6109
mvalenti@wvu.edu

Tutorial Overview

- | | |
|---|--------------------------|
| ■ Channel capacity | 1:15 PM
Valenti |
| ■ Convolutional codes <ul style="list-style-type: none">– the MAP algorithm | |
| ■ Turbo codes <ul style="list-style-type: none">– Standard binary turbo codes: UMTS and cdma2000– Duobinary CRSC turbo codes: DVB-RCS and 802.16 | |
| ■ LDPC codes <ul style="list-style-type: none">– Tanner graphs and the message passing algorithm– Standard binary LDPC codes: DVB-S2 | 3:15 PM
Iyer Seshadri |
| ■ Bit interleaved coded modulation (BICM) <ul style="list-style-type: none">– Combining high-order modulation with a binary capacity approaching code. | 4:30 PM
Valenti |
| ■ EXIT chart analysis of turbo codes | |

Software to Accompany Tutorial

- Iterative Solution's Coded Modulation Library (CML) is a library for simulating and analyzing coded modulation.
- Available for free at the **Iterative Solutions** website:
 - www.iterativesolutions.com
- Runs in matlab, but uses c-mex for efficiency.
- Supported features:
 - Simulation of BICM
 - Turbo, LDPC, or convolutional codes.
 - PSK, QAM, FSK modulation.
 - BICM-ID: Iterative demodulation and decoding.
 - Generation of ergodic capacity curves (BICM/CM constraints).
 - Information outage probability in block fading.
 - Calculation of throughput of hybrid-ARQ.
- Implemented standards:
 - Binary turbo codes: UMTS/3GPP, cdma2000/3GPP2.
 - Duobinary turbo codes: DVB-RCS, wimax/802.16.
 - LDPC codes: DVB-S2.

6/7/2006

Turbo and LDPC Codes

3/133

Noisy Channel Coding Theorem

- Claude Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, 1948.
- Every channel has associated with it a **capacity** C .
 - Measured in bits per channel use (modulated symbol).
- The channel capacity is an upper bound on **information rate** r .
 - There exists a code of rate $r < C$ that achieves reliable communications.
 - Reliable means an arbitrarily small error probability.

6/7/2006

Turbo and LDPC Codes

4/133

Computing Channel Capacity

- The capacity is the **mutual information** between the channel's input X and output Y maximized over all possible input distributions:

$$C = \max_{p(x)} \{I(X; Y)\}$$
$$= \max_{p(x)} \left\{ \iint p(x, y) \log_2 \frac{p(x, y)}{p(x)p(y)} dx dy \right\}$$

6/7/2006

Turbo and LDPC Codes

5/133

Capacity of AWGN with Unconstrained Input

- Consider an AWGN channel with 1-dimensional input:
 - $y = x + n$
 - where n is Gaussian with variance $N_o/2$
 - x is a signal with average energy (variance) E_s

- The capacity in this channel is:

$$C = \max_{p(x)} \{I(X; Y)\} = \frac{1}{2} \log_2 \left(\frac{2E_s}{N_o} + 1 \right) = \frac{1}{2} \log_2 \left(\frac{2rE_b}{N_o} + 1 \right)$$

- where E_b is the energy per (information) bit.

- This capacity is achieved by a Gaussian input x.
 - This is not a practical modulation.

6/7/2006

Turbo and LDPC Codes

6/133

Capacity of AWGN with BPSK Constrained Input

- If we only consider antipodal (BPSK) modulation, then

$$X = \pm\sqrt{E_s}$$

- and the capacity is:

$$\begin{aligned} C &= \max_{p(x)} \{I(X; Y)\} \\ &= I(X; Y) \Big|_{p(x): p=1/2} \quad \text{maximized when two signals are equally likely} \\ &= H(Y) - H(N) \\ &= \int_{-\infty}^{\infty} p(y) \log_2 p(y) dy - \frac{1}{2} \log_2 (\pi e N_o) \end{aligned}$$

This term must be integrated numerically with

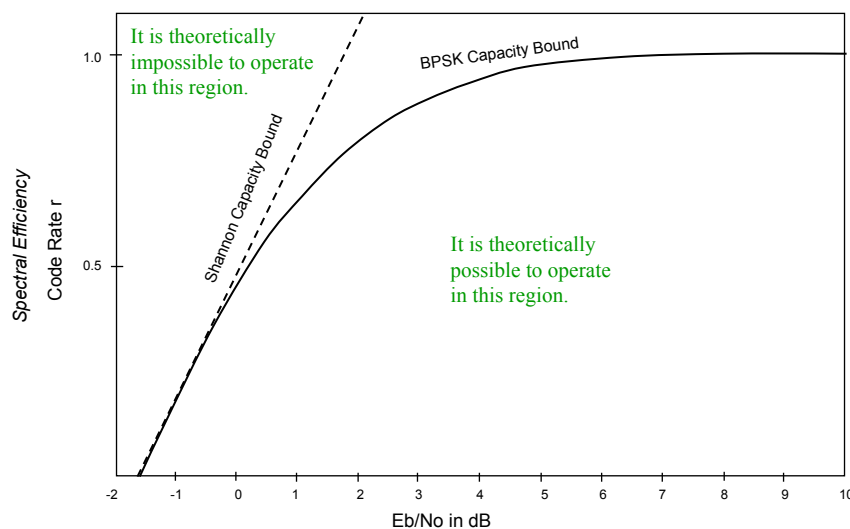
$$p_Y(y) = p_X(y) * p_N(y) = \int_{-\infty}^{\infty} p_X(\lambda) p_N(y - \lambda) d\lambda$$

6/7/2006

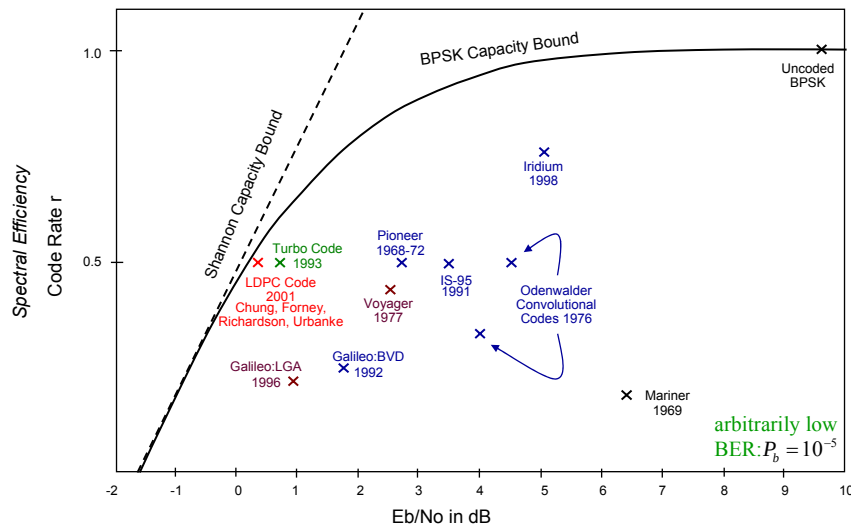
Turbo and LDPC Codes

7/133

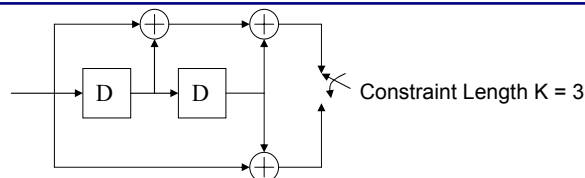
Capacity of AWGN w/ 1-D Signaling



Power Efficiency of Standard Binary Channel Codes



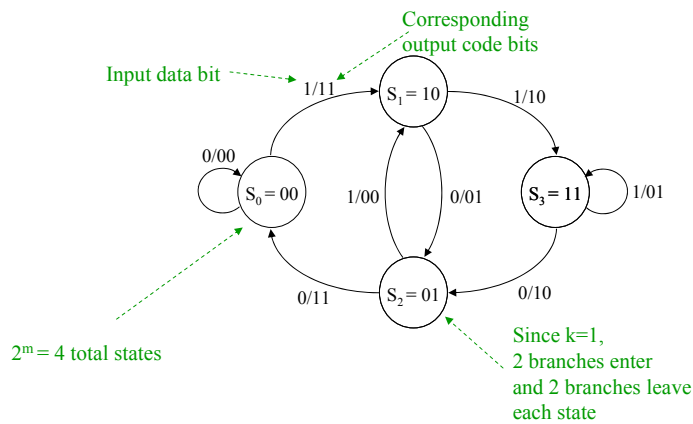
Binary Convolutional Codes



- A **convolutional encoder** comprises:
 - k input streams
 - We assume $k=1$ throughout this tutorial.
 - n output streams
 - m delay elements arranged in a shift register.
 - Combinatorial logic (OR gates).
 - Each of the n outputs depends on some modulo-2 combination of the k current inputs and the m previous inputs in storage
- The **constraint length** is the maximum number of past and present input bits that each output bit can depend on.
 - $K = m + 1$

State Diagrams

- A convolutional encoder is a finite state machine, and can be represented in terms of a state diagram.



6/7/2006

Turbo and LDPC Codes

11/133

Trellis Diagram

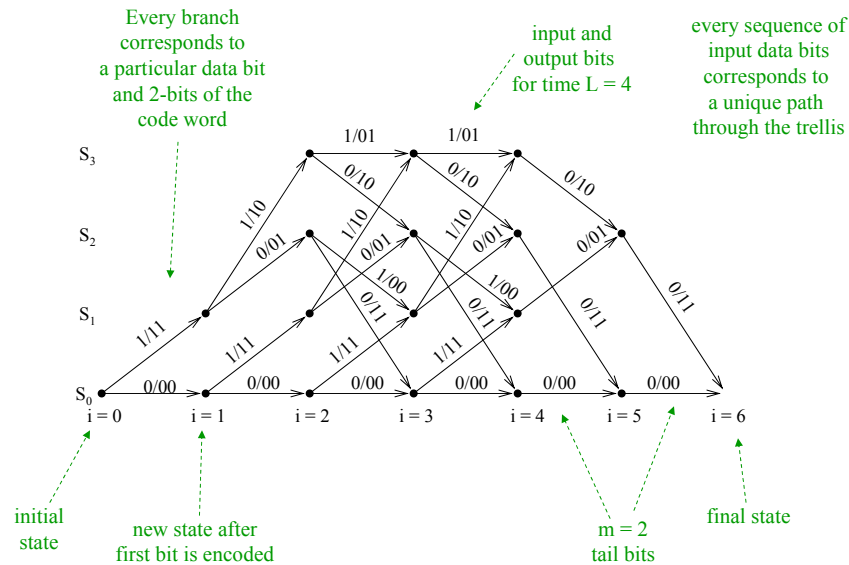
- Although a state diagram is a helpful tool to understand the operation of the encoder, it does not show how the states change over time for a particular input sequence.
- A **trellis** is an expansion of the state diagram which explicitly shows the passage of time.
 - All the possible states are shown for each instant of time.
 - Time is indicated by a movement to the right.
 - The input data bits and output code bits are represented by a unique path through the trellis.

6/7/2006

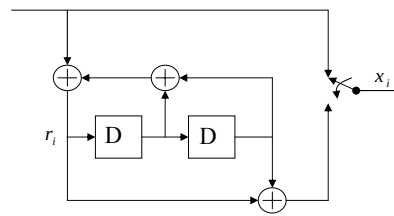
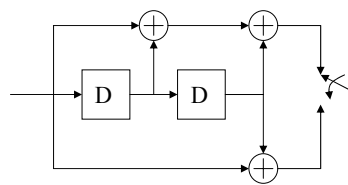
Turbo and LDPC Codes

12/133

Trellis Diagram



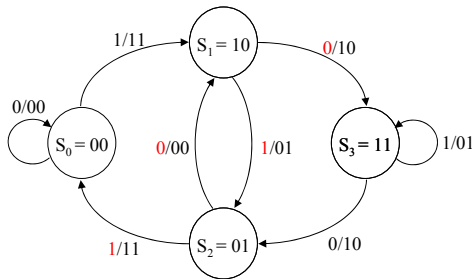
Recursive Systematic Convolutional (RSC) Codes



- An **RSC** encoder is constructed from a standard convolutional encoder by feeding back one of the outputs.
- An RSC code is **systematic**.
 - The input bits appear directly in the output.
- An RSC encoder is an **Infinite Impulse Response (IIR)** Filter.
 - An arbitrary input will cause a “good” (high weight) output with high probability.
 - Some inputs will cause “bad” (low weight) outputs.

State Diagram of RSC Code

- With an RSC code, the output labels are the same.
- However, input labels are changed so that each state has an input “0” and an input “1”



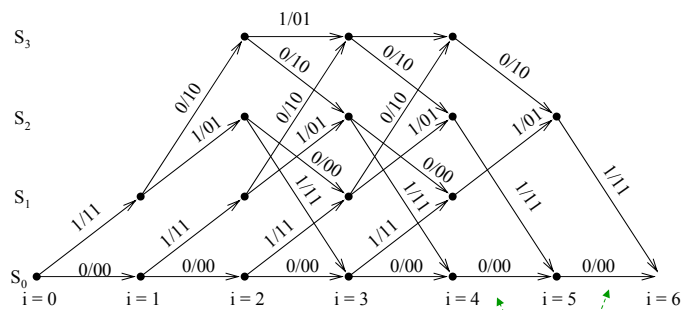
- Messages labeling transitions that start from S_1 and S_2 are complemented.

6/7/2006

Turbo and LDPC Codes

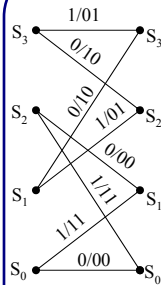
15/133

Trellis Diagram of RSC Code



m = 2 tail bits
no longer all-zeros
must be calculated by the encoder

Convolutional Codewords



- Consider the trellis section at time t .
 - Let $S(t)$ be the encoder state at time t .
 - When there are four states, $S(t) \in \{S_0, S_1, S_2, S_3\}$
- Let $u(t)$ be the message bit at time t .
 - The encoder state $S(t)$ depends on $u(t)$ and $S(t-1)$
- Depending on its initial state $S(t-1)$ and the final state $S(t)$, the encoder will generate an n -bit long word
 - $\mathbf{x}(t) = (x_1, x_2, \dots, x_n)$
- The word is transmitted over a channel during time t , and the received signal is:
 - $\mathbf{y}(t) = (y_1, y_2, \dots, y_n)$
 - For BPSK, each $y = (2x-1) + n$
- If there are L input data bits plus m tail bits, the overall transmitted codeword is:
 - $\mathbf{x} = [\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(L), \dots, \mathbf{x}(L+m)]$
- And the received codeword is:
 - $\mathbf{y} = [\mathbf{y}(1), \mathbf{y}(2), \dots, \mathbf{y}(L), \dots, \mathbf{y}(L+m)]$

6/7/2006

Turbo and LDPC Codes

17/133

MAP Decoding

- The goal of the maximum a posteriori (MAP) decoder is to determine $P(u(t)=1 | \mathbf{y})$ and $P(u(t)=0 | \mathbf{y})$ for each t .
 - The probability of each message bit, given the entire received codeword.
- These two probabilities are conveniently expressed as a log-likelihood ratio:

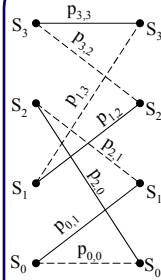
$$\lambda(t) = \log \frac{P[u(t)=1 | \mathbf{y}]}{P[u(t)=0 | \mathbf{y}]}$$

6/7/2006

Turbo and LDPC Codes

18/133

Determining Message Bit Probabilities from the Branch Probabilities



- Let $p_{i,j}(t)$ be the probability that the encoder made a transition from S_i to S_j at time t , given the entire received codeword.

- $p_{i,j}(t) = P(S_i(t-1) \rightarrow S_j(t) | \mathbf{y})$
 - where $S_j(t)$ means that $S(t)=S_j$

- For each t ,

$$\sum_{S_i \rightarrow S_j} P(S_i(t-1) \rightarrow S_j(t) | \mathbf{y}) = 1$$

- The probability that $u(t) = 1$ is

$$P(u(t) = 1 | \mathbf{y}) = \sum_{S_i \rightarrow S_j, u=1} P(S_i(t-1) \rightarrow S_j(t) | \mathbf{y})$$

- Likewise

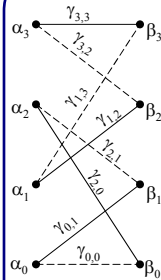
$$P(u(t) = 0 | \mathbf{y}) = \sum_{S_i \rightarrow S_j, u=0} P(S_i(t-1) \rightarrow S_j(t) | \mathbf{y})$$

6/7/2006

Turbo and LDPC Codes

19/133

Determining the Branch Probabilities



- Let $\gamma_{i,j}(t)$ = Probability of transition from state S_i to state S_j at time t , given just the received word $\mathbf{y}(t)$

- $\gamma_{i,j}(t) = P(S_i(t-1) \rightarrow S_j(t) | \mathbf{y}(t))$

- Let $\alpha_i(t-1)$ = Probability of starting at state S_i at time t , given all symbols received prior to time t .

- $\alpha_i(t-1) = P(S_i(t-1) | \mathbf{y}(1), \mathbf{y}(2), \dots, \mathbf{y}(t-1))$

- β_j = Probability of ending at state S_j at time t , given all symbols received after time t .

- $\beta_j(t) = P(S_j(t) | \mathbf{y}(t+1), \dots, \mathbf{y}(L+m))$

- Then the branch probability is:

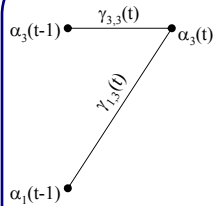
- $p_{i,j}(t) = \alpha_i(t-1) \gamma_{i,j}(t) \beta_j(t)$

6/7/2006

Turbo and LDPC Codes

20/133

Computing α



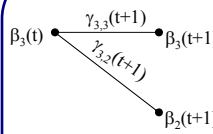
- α can be computed recursively.
- Prob. of path going through $S_i(t-1)$ and terminating at $S_j(t)$, given $\mathbf{y}(1) \dots \mathbf{y}(t)$ is:
 - $\alpha_i(t-1) \gamma_{ij}(t)$
- Prob. of being in state $S_j(t)$, given $\mathbf{y}(1) \dots \mathbf{y}(t)$ is found by adding the probabilities of the two paths terminating at state $S_j(t)$.
- For example,
 - $\alpha_3(t) = \alpha_1(t-1) \gamma_{1,3}(t) + \alpha_3(t-1) \gamma_{3,3}(t)$
- The values of α can be computed for every state in the trellis by “sweeping” through the trellis in the **forward** direction.

6/7/2006

Turbo and LDPC Codes

21/133

Computing β



- Likewise, β is computed recursively.
- Prob. of path going through $S_j(t+1)$ and terminating at $S_i(t)$, given $\mathbf{y}(t+1), \dots, \mathbf{y}(L+m)$
 - $\beta_j(t+1) \gamma_{ij}(t+1)$
- Prob. of being in state $S_i(t)$, given $\mathbf{y}(t+1), \dots, \mathbf{y}(L+m)$ is found by adding the probabilities of the two paths starting at state $S_i(t)$.
- For example,
 - $\beta_3(t) = \beta_2(t+1) \gamma_{1,2}(t+1) + \beta_3(t+1) \gamma_{3,3}(t+1)$
- The values of β can be computed for every state in the trellis by “sweeping” through the trellis in the **reverse** direction.

6/7/2006

Turbo and LDPC Codes

22/133

Computing γ

- Every branch in the trellis is labeled with:
 - $\gamma_{i,j}(t) = P(S_i(t-1) \rightarrow S_j(t) | \mathbf{y}(t))$
- Let $\mathbf{x}_{i,j} = (x_1, x_2, \dots, x_n)$ be the word generated by the encoder when transitioning from S_i to S_j .
 - $\gamma_{i,j}(t) = P(\mathbf{x}_{i,j} | \mathbf{y}(t))$
- From Bayes rule,
 - $\gamma_{i,j}(t) = P(\mathbf{x}_{i,j} | \mathbf{y}(t)) = P(\mathbf{y}(t) | \mathbf{x}_{i,j}) P(\mathbf{x}_{i,j}) / P(\mathbf{y}(t))$
- $P(\mathbf{y}(t))$
 - Is not strictly needed because will be the same value for the numerator and denominator of the LLR $\lambda(t)$.
 - Instead of computing directly, can be found indirectly as a normalization factor (chosen for numerical stability)
- $P(\mathbf{x}_{i,j})$
 - Initially found assuming that code bits are equally likely.
 - In a turbo code, this is provided to the decoder as “a priori” information.

6/7/2006

Turbo and LDPC Codes

23/133

Computing $P(\mathbf{y}(t) | \mathbf{x}_{i,j})$

- If BPSK modulation is used over an AWGN channel, the probability of code **bit** y given x is conditionally Gaussian:

$$P(y | x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(y - m_x)^2}{2\sigma^2}\right\}$$

$$m_x = \sqrt{E_s}(2x - 1)$$

$$\sigma^2 = \frac{N_0}{2}$$

- In Rayleigh fading, multiply m_x by a , the fading amplitude.

- The conditional probability of the **word** $\mathbf{y}(t)$

$$P(\mathbf{y} | \mathbf{x}) = \prod_{i=1}^n p(y_i | x_i)$$

6/7/2006

Turbo and LDPC Codes

24/133

Overview of MAP algorithm

- Label every branch of the trellis with $\gamma_{i,j}(t)$.
- Sweep through trellis in forward-direction to compute $\alpha_i(t)$ at every node in the trellis.
- Sweep through trellis in reverse-direction to compute $\beta_j(t)$ at every node in the trellis.
- Compute the LLR of the message bit at each trellis section:

$$\begin{aligned}\lambda(t) &= \log \frac{P[u(t) = 1 | \mathbf{y}]}{P[u(t) = 0 | \mathbf{y}]} \\ &= \log \frac{\sum_{S_i \rightarrow S_j: u=1} \alpha_i(t-1) \gamma_{i,j}(t) \beta_j(t)}{\sum_{S_i \rightarrow S_j: u=0} \alpha_i(t-1) \gamma_{i,j}(t) \beta_j(t)}\end{aligned}$$

- MAP algorithm also called the “forward-backward” algorithm (Forney).

6/7/2006

Turbo and LDPC Codes

25/133

Log Domain Decoding

- The MAP algorithm can be simplified by performing in the log domain.
 - exponential terms (e.g. used to compute γ) disappear.
 - multiplications become additions.
 - Addition can be approximated with maximization.
- Redefine all quantities:
 - $\gamma_{i,j}(t) = \log P(S_i(t-1) \rightarrow S_j(t) | \mathbf{y}(t))$
 - $\alpha_i(t-1) = \log P(S_i(t-1) | \mathbf{y}(1), \mathbf{y}(2), \dots, \mathbf{y}(t-1))$
 - $\beta_j(t) = \log P(S_j(t) | \mathbf{y}(t+1), \dots, \mathbf{y}(L+m))$
- Details of the log-domain implementation will be presented later...

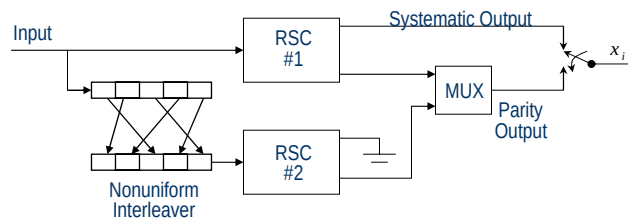
6/7/2006

Turbo and LDPC Codes

26/133

Parallel Concatenated Codes with Nonuniform Interleaving

- A stronger code can be created by encoding in parallel.
- A **nonuniform interleaver** scrambles the ordering of bits at the input of the second encoder.
 - Uses a pseudo-random interleaving pattern.
- It is very unlikely that both encoders produce low weight code words.
- MUX increases code rate from 1/3 to 1/2.



6/7/2006

Turbo and LDPC Codes

27/133

Random Coding Interpretation of Turbo Codes

- **Random codes** achieve the best performance.
 - Shannon showed that as $n \rightarrow \infty$, random codes achieve channel capacity.
- However, random codes are not feasible.
 - The code must contain enough structure so that decoding can be realized with actual hardware.
- **Coding dilemma:**
 - “All codes are good, except those that we can think of.”
- With turbo codes:
 - The nonuniform interleaver adds *apparent* randomness to the code.
 - Yet, they contain enough structure so that decoding is feasible.

6/7/2006

Turbo and LDPC Codes

28/133

Comparison of a Turbo Code and a Convolutional Code

- First consider a K=12 convolutional code.
 - $d_{\min} = 18$
 - $\beta_d = 187$ (output weight of all $d_{\min}$ paths)
- Now consider the original turbo code.
 - C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in Proc. *IEEE Int. Conf. on Commun.*, Geneva, Switzerland, May 1993, pp. 1064-1070.
 - Same complexity as the K=12 convolutional code
 - Constraint length 5 RSC encoders
 - $k = 65,536$ bit interleaver
 - Minimum distance $d_{\min} = 6$
 - $a_d = 3$ minimum distance code words
 - Minimum distance code words have average information weight of only

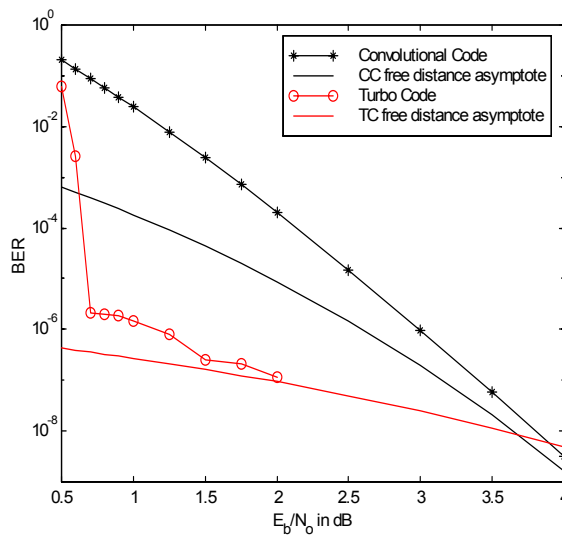
$$f_d = 2$$

6/7/2006

Turbo and LDPC Codes

29/133

Comparison of Minimum-distance Asymptotes



- Convolutional code:

$$d_{\min} = 18$$

$$c_{d\min} = 187$$

$$P_b \approx (187)Q\left(\sqrt{18\frac{E_b}{N_o}}\right)$$

- Turbo code:

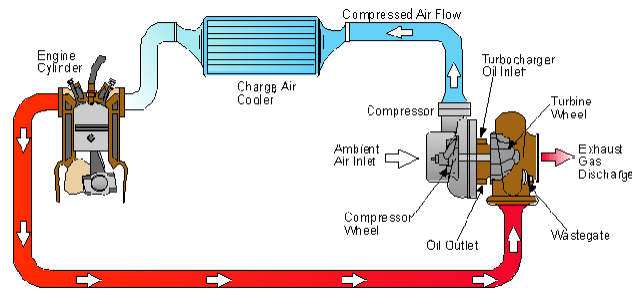
$$d_{\min} = 6$$

$$\tilde{c}_{d\min} = \frac{a_{d\min} \tilde{w}_{d\min}}{k} = \frac{3 \cdot 2}{65536}$$

$$P_b \approx (9.2 \times 10^{-5})Q\left(\sqrt{6\frac{E_b}{N_o}}\right)$$

The Turbo-Principle

- Turbo codes get their name because the decoder uses feedback, like a turbo engine.

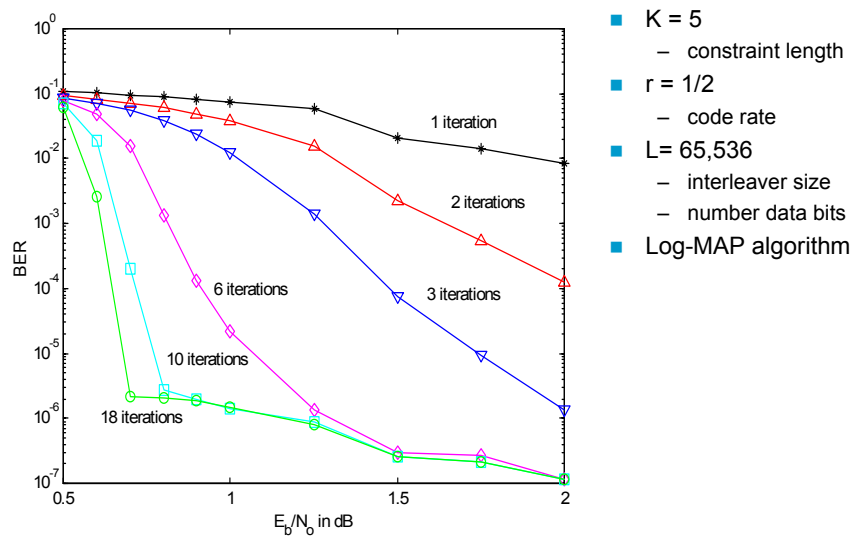


Turbo and LDPC Codes

6/7/2006

31/133

Performance as a Function of Number of Iterations



Summary of Performance Factors and Tradeoffs

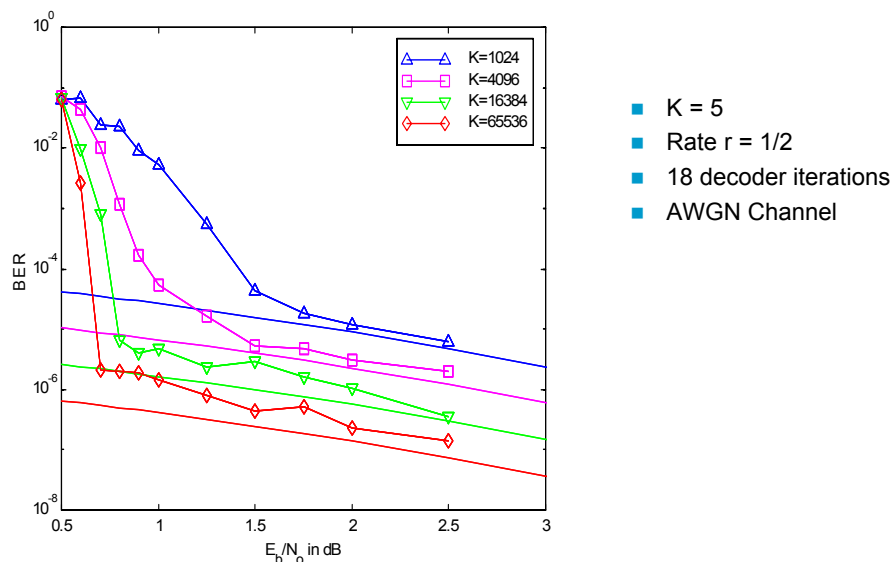
- Latency vs. performance
 - Frame (interleaver) size L
- Complexity vs. performance
 - Decoding algorithm
 - Number of iterations
 - Encoder constraint length K
- Spectral efficiency vs. performance
 - Overall code rate r
- Other factors
 - Interleaver design
 - Puncture pattern
 - Trellis termination

6/7/2006

Turbo and LDPC Codes

33/133

Tradeoff: BER Performance versus Frame Size (Latency)



Characteristics of Turbo Codes

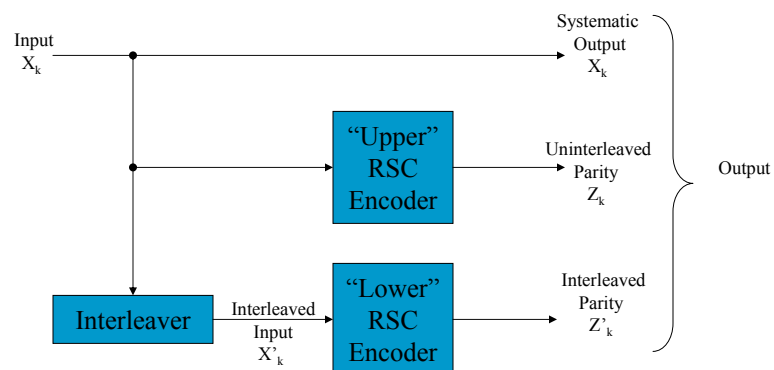
- Turbo codes have extraordinary performance at low SNR.
 - Very close to the Shannon limit.
 - Due to a low multiplicity of low weight code words.
- However, turbo codes have a BER “floor”.
 - This is due to their low minimum distance.
- Performance improves for larger block sizes.
 - Larger block sizes mean more latency (delay).
 - However, larger block sizes are not more complex to decode.
 - The BER floor is lower for larger frame/interleaver sizes
- The complexity of a constraint length K_{TC} turbo code is the same as a $K = K_{CC}$ convolutional code, where:
 - $K_{CC} \approx 2 + K_{TC} + \log_2(\text{number decoder iterations})$

6/7/2006

Turbo and LDPC Codes

35/133

UMTS Turbo Encoder



- From 3GPP TS 25 212 v6.6.0, Release 6 (2005-09)
 - UMTS Multiplexing and channel coding
- Data is segmented into blocks of L bits.
 - where $40 \leq L \leq 5114$

UMTS Interleaver: Inserting Data into Matrix

- Data is fed row-wise into a R by C matrix.
 - R = 5, 10, or 20.
 - $8 \leq C \leq 256$
 - If $L < RC$ then matrix is padded with dummy characters.

In the CML, the UMTS interleaver is created by the function **CreateUMTSInterleaver**.
Interleaving and Deinterleaving are implemented by **Interleave** and **Deinterleave**.

X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	→
X ₉	X ₁₀	X ₁₁	X ₁₂	X ₁₃	X ₁₄	X ₁₅	X ₁₆	→
X ₁₇	X ₁₈	X ₁₉	X ₂₀	X ₂₁	X ₂₂	X ₂₃	X ₂₄	→
X ₂₅	X ₂₆	X ₂₇	X ₂₈	X ₂₉	X ₃₀	X ₃₁	X ₃₂	→
X ₃₃	X ₃₄	X ₃₅	X ₃₆	X ₃₇	X ₃₈	X ₃₉	X ₄₀	→

6/7/2006

Turbo and LDPC Codes

37/133

UMTS Interleaver: Intra-Row Permutations

- Data is permuted *within* each row.
 - Permutation rules are rather complicated.
 - See spec for details.

X ₂	X ₆	X ₅	X ₇	X ₃	X ₄	X ₁	X ₈
X ₁₀	X ₁₂	X ₁₁	X ₁₅	X ₁₃	X ₁₄	X ₉	X ₁₆
X ₁₈	X ₂₂	X ₂₁	X ₂₃	X ₁₉	X ₂₀	X ₁₇	X ₂₄
X ₂₆	X ₂₈	X ₂₇	X ₃₁	X ₂₉	X ₃₀	X ₂₅	X ₃₂
X ₄₀	X ₃₆	X ₃₅	X ₃₉	X ₃₇	X ₃₈	X ₃₃	X ₃₄

6/7/2006

Turbo and LDPC Codes

38/133

UMTS Interleaver: Inter-Row Permutations

- Rows are permuted.
 - If $R = 5$ or 10 , the matrix is reflected about the middle row.
 - For $R=20$ the rule is more complicated and depends on L .
 - See spec for $R=20$ case.

X_{40}	X_{36}	X_{35}	X_{39}	X_{37}	X_{38}	X_{33}	X_{34}
X_{26}	X_{28}	X_{27}	X_{31}	X_{29}	X_{30}	X_{25}	X_{32}
X_{18}	X_{22}	X_{21}	X_{23}	X_{19}	X_{20}	X_{17}	X_{24}
X_{10}	X_{12}	X_{11}	X_{15}	X_{13}	X_{14}	X_9	X_{16}
X_2	X_6	X_5	X_7	X_3	X_4	X_1	X_8

6/7/2006

Turbo and LDPC Codes

39/133

UMTS Interleaver: Reading Data From Matrix

- Data is read from matrix column-wise.

X_{40}	X_{36}	X_{35}	X_{39}	X_{37}	X_{38}	X_{33}	X_{34}
X_{26}	X_{28}	X_{27}	X_{31}	X_{29}	X_{30}	X_{25}	X_{32}
X_{18}	X_{22}	X_{21}	X_{23}	X_{19}	X_{20}	X_{17}	X_{24}
X_{10}	X_{12}	X_{11}	X_{15}	X_{13}	X_{14}	X_9	X_{16}
X_2	X_6	X_5	X_7	X_3	X_4	X_1	X_8

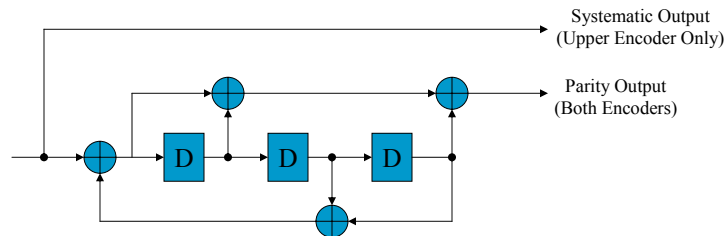
- Thus:
 - $X'_1 = X_{40}$ $X'_2 = X_{26}$ $X'_3 = X_{18}$...
 - $X'_{38} = X_{24}$ $X'_{39} = X_{16}$ $X'_{40} = X_8$

6/7/2006

Turbo and LDPC Codes

40/133

UMTS Constituent RSC Encoder



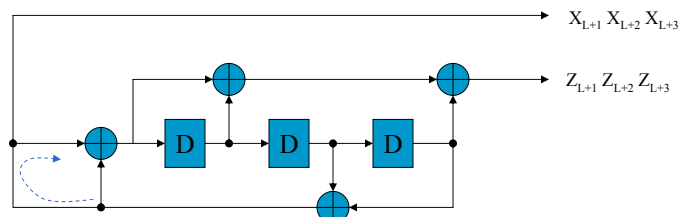
- Upper and lower encoders are identical:
 - Feedforward generator is 15 in octal.
 - Feedback generator is 13 in octal.

6/7/2006

Turbo and LDPC Codes

41/133

Trellis Termination



- After the L^{th} input bit, a 3 bit tail is calculated.
 - The tail bit equals the fed back bit.
 - This guarantees that the registers get filled with zeros.
- Each encoder has its own tail.
 - The tail bits and their parity bits are transmitted at the end.

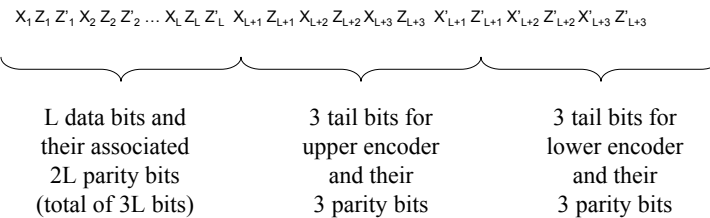
6/7/2006

Turbo and LDPC Codes

42/133

Output Stream Format

- The format of the output stream is:



Total number of coded bits = $3L + 12$

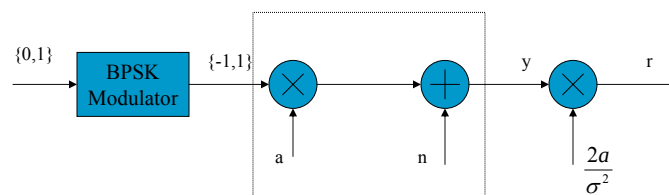
$$\text{Code rate: } r = \frac{L}{3L+12} \approx \frac{1}{3}$$

6/7/2006

Turbo and LDPC Codes

43/133

Channel Model and LLRs



- Channel gain: a
 - Rayleigh random variable if Rayleigh fading
 - $a = 1$ if AWGN channel
- Noise
 - variance is: $\sigma^2 = \frac{1}{2r \left(\frac{E_b}{N_o} \right)} \approx \frac{3}{2 \left(\frac{E_b}{N_o} \right)}$

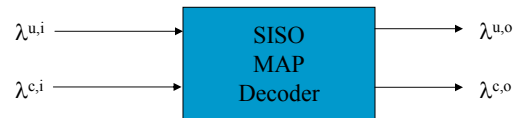
6/7/2006

Turbo and LDPC Codes

44/133

SISO-MAP Decoding Block

This block is implemented in the CML by the `SisoDecode` function



Inputs:

- $\lambda^{u,i}$ LLR's of the data bits. This comes from the other decoder r .
- $\lambda^{c,i}$ LLR's of the code bits. This comes from the channel observations r .

Two output streams:

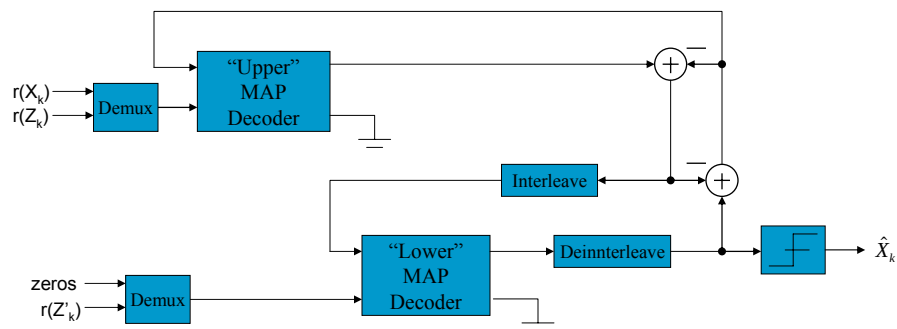
- $\lambda^{u,o}$ LLR's of the data bits. Passed to the other decoder.
- $\lambda^{c,o}$ LLR's of the code bits. Not used by the other decoder.

6/7/2006

Turbo and LDPC Codes

45/133

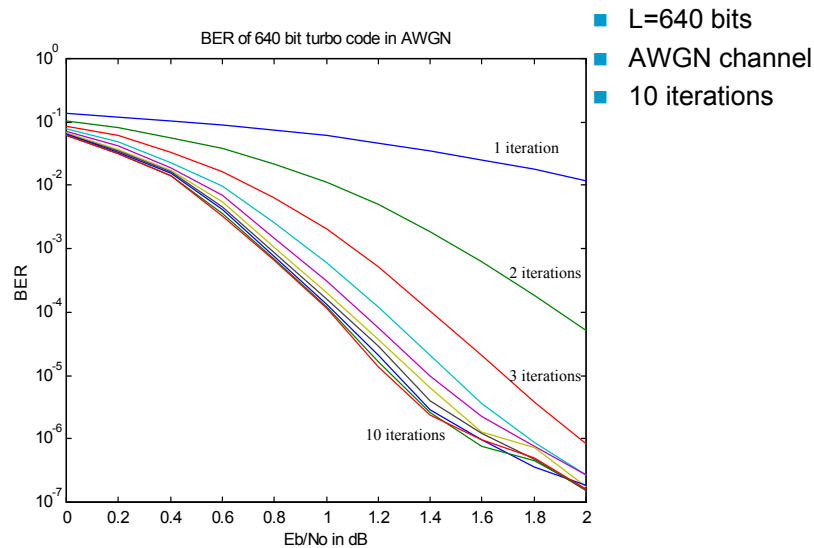
Turbo Decoding Architecture



Initialization and timing:

- Upper $\lambda^{u,i}$ input is initialized to all zeros.
- Upper decoder executes first, then lower decoder.

Performance as a Function of Number of Iterations



Log-MAP Algorithm: Overview

- Log-MAP algorithm is MAP implemented in log-domain.
 - Multiplications become additions.
 - Additions become special “max*” operator (Jacobi logarithm)
- Log-MAP is similar to the Viterbi algorithm.
 - Except “max” is replaced by “max*” in the ACS operation.
- Processing:
 - Sweep through the trellis in **forward** direction using modified Viterbi algorithm.
 - Sweep through the trellis in **backward** direction using modified Viterbi algorithm.
 - Determine LLR for each trellis section.
 - Determine output extrinsic info for each trellis section.

The max* operator

- max* must implement the following operation:

$$\begin{aligned} z &= \max(x, y) + \ln(1 + \exp\{-|y - x|\}) \\ &= \max(x, y) + f_c(|y - x|) \\ &= \max^*(x, y) \end{aligned}$$

- Ways to accomplish this:

- C-function calls or large look-up-table. log-MAP
- (Piecewise) linear approximation.
- Rough correction value.

$$z \approx \max(x, y) + \begin{cases} 0 & \text{if } |y - x| > 1.5 \\ 0.5 & \text{if } |y - x| \leq 1.5 \end{cases} \quad \text{constant-log-MAP}$$

- Max operator.

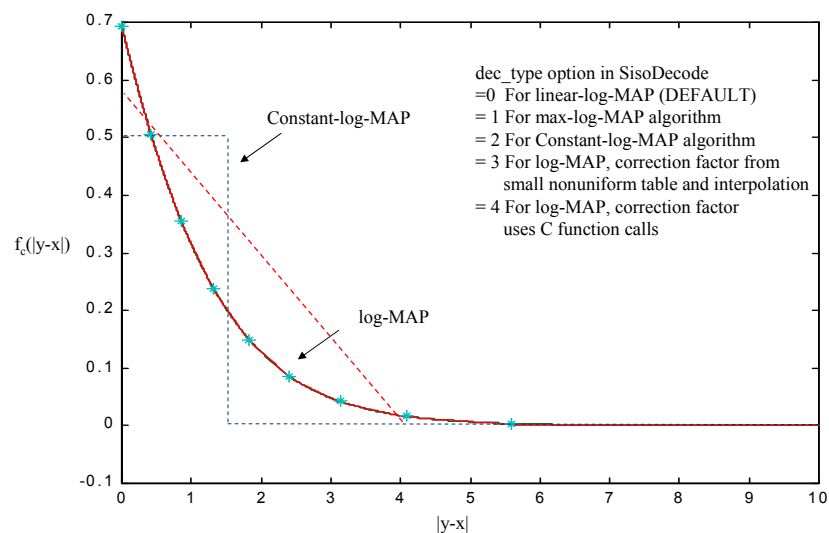
$$z \approx \max(x, y) \quad \text{max-log-MAP}$$

6/7/2006

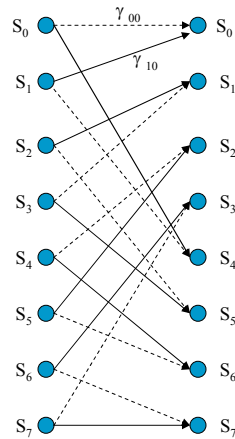
Turbo and LDPC Codes

49/133

The Correction Function



The Trellis for UMTS



- Dotted line = data 0
- Solid line = data 1
- Note that each node has one each of data 0 and 1 entering and leaving it.
- The branch from node S_i to S_j has metric γ_{ij}

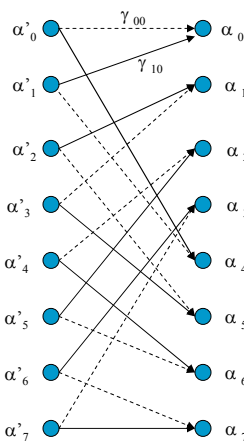
$$\gamma_{ij} = \underbrace{X_k(i, j)\lambda_k^{u,i}}_{\text{data bit associated with branch } S_i \rightarrow S_j} + \underbrace{X_k(i, j)\lambda_{k_1}^{c,i}}_{\text{The two code bits labeling with branch } S_i \rightarrow S_j} + \underbrace{Z_k(i, j)\lambda_{k_2}^{c,i}}_{\text{The two code bits labeling with branch } S_i \rightarrow S_j}$$

6/7/2006

Turbo and LDPC Codes

51/133

Forward Recursion



- A new metric must be calculated for each node in the trellis using:

$$\alpha_j = \max^* \{ (\alpha'_{i_1} + \gamma_{i_1 j}), (\alpha'_{i_2} + \gamma_{i_2 j}) \}$$

- where i_1 and i_2 are the two states connected to j .
- Start from the beginning of the trellis (i.e. the left edge).
- Initialize stage 0:

$$\alpha_0 = 0$$

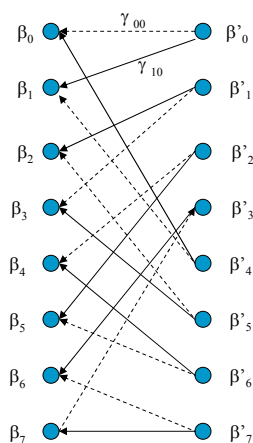
$$\alpha_i = -\infty \text{ for all } i \neq 0$$

6/7/2006

Turbo and LDPC Codes

52/133

Backward Recursion



- A new metric must be calculated for each node in the trellis using:

$$\beta_i = \max^* \{ (\beta'_{j_1} + \gamma_{ij_1}), (\beta'_{j_2} + \gamma_{ij_2}) \}$$

- where j_1 and j_2 are the two states connected to i .
- Start from the end of the trellis (i.e. the right edge).
- Initialize stage $L+3$:

$$\beta_0 = 0$$

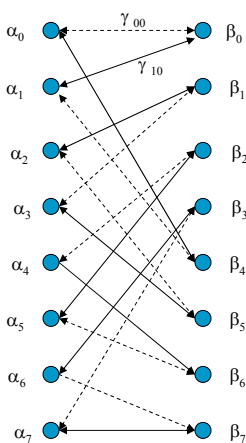
$$\beta_i = -\infty \text{ for all } i \neq 0$$

6/7/2006

Turbo and LDPC Codes

53/133

Log-likelihood Ratio



- The likelihood of any one branch is:

$$\alpha_i + \gamma_{ij} + \beta_j$$

- The likelihood of data 1 is found by summing the likelihoods of the solid branches.

- The likelihood of data 0 is found by summing the likelihoods of the dashed branches.

- The log likelihood ratio (LLR) is:

$$\begin{aligned} \Lambda(X_k) &= \ln \left(\frac{P[X_k = 1]}{P[X_k = 0]} \right) \\ &= \max_{S_i \rightarrow S_j: X_k=1}^* \{ \alpha_i + \gamma_{ij} + \beta_j \} \\ &\quad - \max_{S_i \rightarrow S_j: X_k=0}^* \{ \alpha_i + \gamma_{ij} + \beta_j \} \end{aligned}$$

6/7/2006

Turbo and LDPC Codes

54/133

Memory Issues

- A naïve solution:
 - Calculate α 's for entire trellis (forward sweep), and store.
 - Calculate β 's for the entire trellis (backward sweep), and store.
 - At the k th stage of the trellis, compute λ by combining γ 's with stored α 's and β 's.
- A better approach:
 - Calculate β 's for the entire trellis and store.
 - Calculate α 's for the k th stage of the trellis, and immediately compute λ by combining γ 's with these α 's and stored β 's.
 - Use the α 's for the k th stage to compute α 's for state $k+1$.
- Normalization:
 - In log-domain, α 's can be normalized by subtracting a common term from all α 's at the same stage.
 - Can normalize relative to α_0 , which eliminates the need to store α_0
 - Same for the β 's

6/7/2006

Turbo and LDPC Codes

55/133

Sliding Window Algorithm

- Can use a sliding window to compute β 's
 - Windows need some overlap due to uncertainty in terminating state.



6/7/2006

Turbo and LDPC Codes

56/133

Extrinsic Information

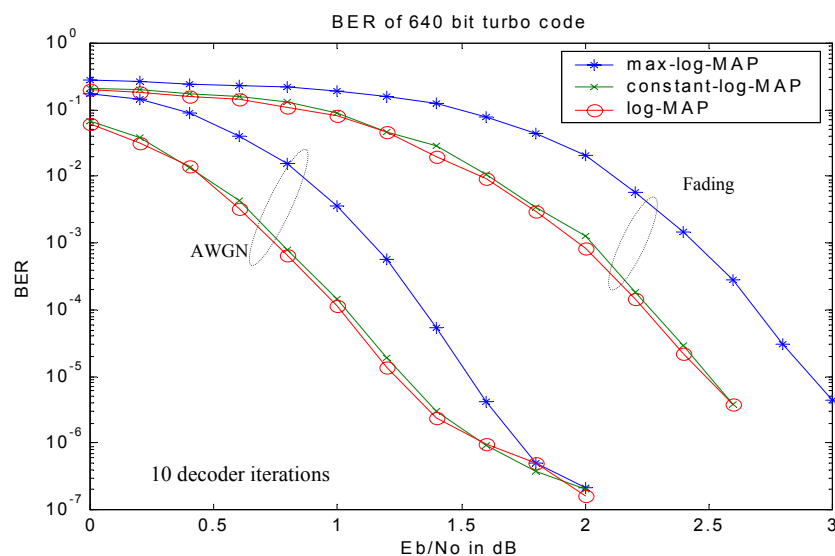
- The extrinsic information is found by subtracting the corresponding input from the LLR output, i.e.
 - $\lambda^{u,i}(\text{lower}) = \lambda^{u,o}(\text{upper}) - \lambda^{u,i}(\text{upper})$
 - $\lambda^{u,i}(\text{upper}) = \lambda^{u,o}(\text{lower}) - \lambda^{u,i}(\text{lower})$
- It is necessary to subtract the information that is already available at the other decoder in order to prevent “positive feedback”.
- The extrinsic information is the amount of new information gained by the current decoder step.

6/7/2006

Turbo and LDPC Codes

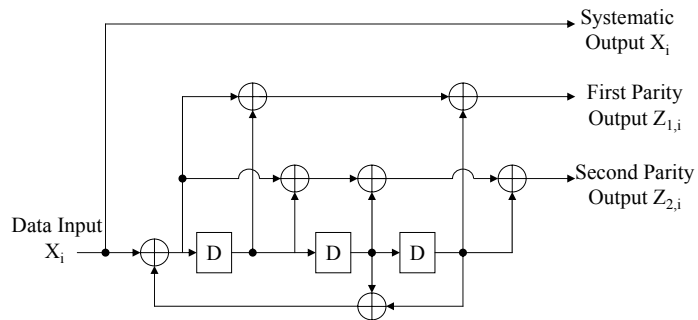
57/133

Performance Comparison



cdma2000

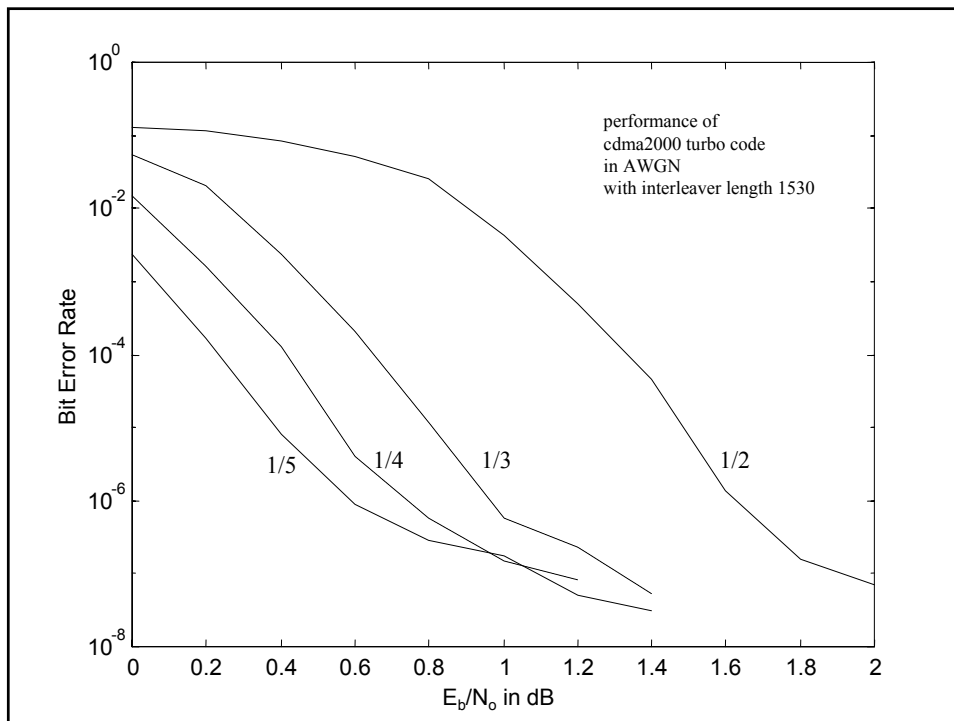
- cdma2000 uses a rate $\frac{1}{3}$ constituent encoder.
 - Overall turbo code rate can be $\frac{1}{5}$, $\frac{1}{4}$, $\frac{1}{3}$, or $\frac{1}{2}$.
 - Fixed interleaver lengths:
 - 378, 570, 762, 1146, 1530, 2398, 3066, 4602, 6138, 9210, 12282, or 20730



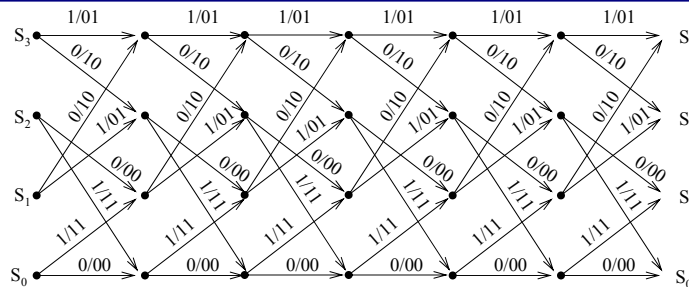
6/7/2006

Turbo and LDPC Codes

59/133



Circular Recursive Systematic Convolutional (CRSC) Codes



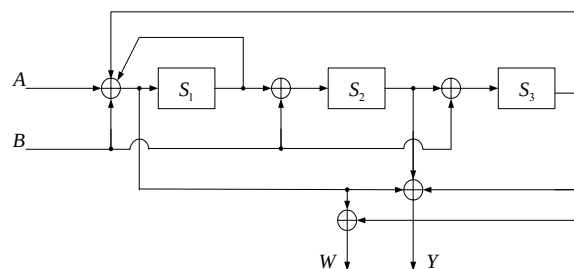
- CRSC codes use the concept of tailbiting.
 - Sequence is encode so that initial state is same as final state.
- Advantage and disadvantages
 - No need for tail bits.
 - Need to encode twice.
 - Complicates decoder.

6/7/2006

Turbo and LDPC Codes

61/133

Duobinary codes



- Duobinary codes are defined over $GF(4)$.
 - two bits taken in per clock cycle.
 - Output is systematic and rate $2/4$.
- Hardware benefits
 - Half as many states in trellis.
 - Smaller loss due to max-log-MAP decoding.

6/7/2006

Turbo and LDPC Codes

62/133

DVB-RCS

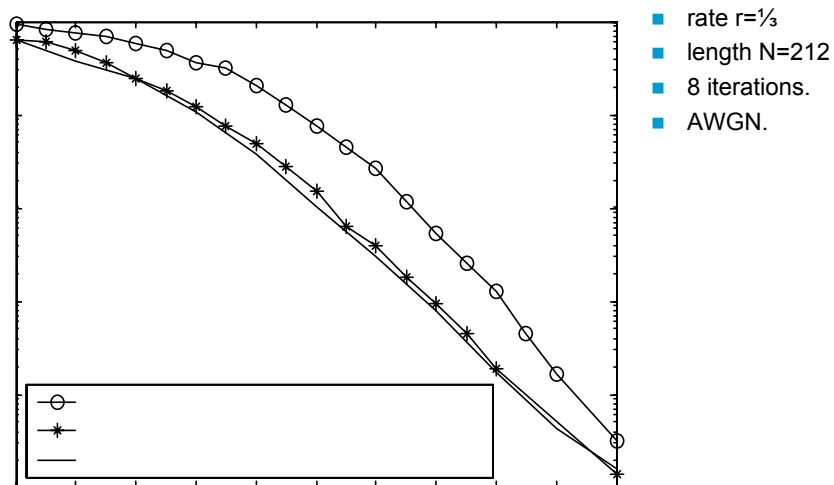
- Digital Video Broadcasting – Return Channel via Satellite.
 - Consumer-grade Internet service over satellite.
 - 144 kbps to 2 Mbps satellite *uplink*.
 - Uses same antenna as downlink.
 - QPSK modulation.
- DVB-RCS uses a pair of duobinary CRSC codes.
- Ket parameters:
 - input of $N = k/2$ couples
 - $N = \{48, 64, 212, 220, 228, 424, 432, 440, 752, 848, 856, 864\}$
 - $r = \{1/3, 2/5, 1/2, 2/3, 3/4, 4/5, 6/7\}$
- M.C. Valenti, S. Cheng, and R. Iyer Seshadri, "Turbo and LDPC codes for digital video broadcasting," Chapter 12 of *Turbo Code Applications: A Journey from a Paper to Realization*, Springer, 2005.

6/7/2006

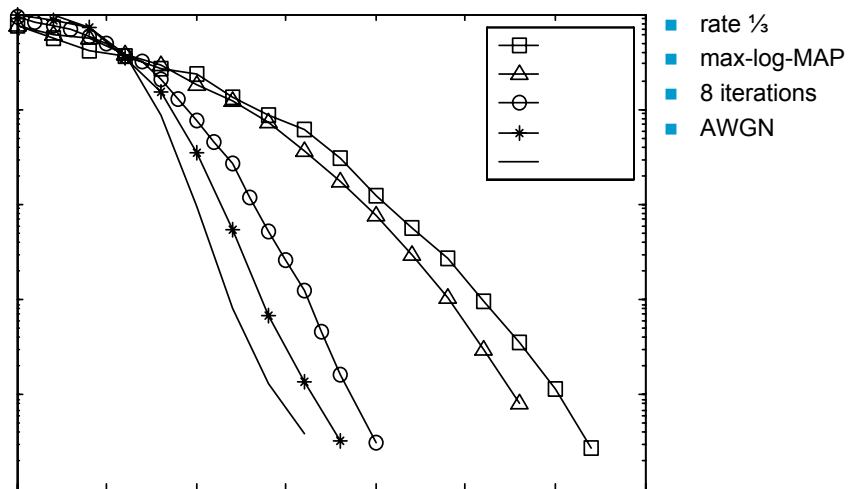
Turbo and LDPC Codes

63/133

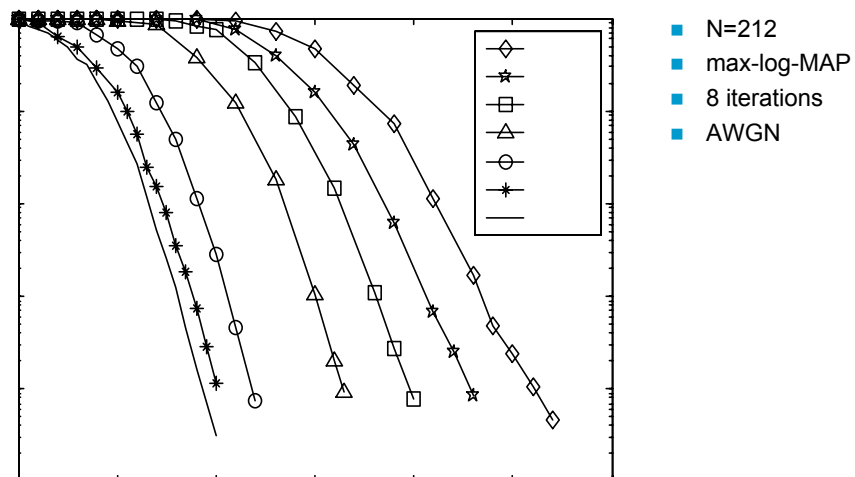
DVB-RCS: Influence of Decoding Algorithm



DVB-RCS: Influence of Block Length



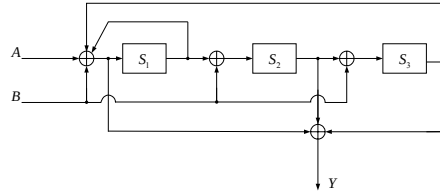
DVB-RCS: Influence of Code Rate



FER
 10^{-3}

802.16 (WiMax)

- The standard specifies an optional convolutional turbo code (CTC) for operation in the 2-11 GHz range.
- Uses same duobinary CRSC encoder as DVB-RCS, though without output W .



- Modulation: BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM.
- Key parameters:
 - Input message size 8 to 256 bytes long.
 - $r = \{1/2, 2/3, 3/4, 5/6, 7/8\}$

6/7/2006

Turbo and LDPC Codes

67/133

Prelude to LDPC Codes: Review of Linear Block Codes

- $V_n = n$ -dimensional vector space over $\{0, 1\}$
- A (n, k) **linear block code** with dataword length k , codeword length n is a k -dimensional vector subspace of V_n
- A codeword $\mathbf{c}$ is generated by the matrix multiplication $\mathbf{c} = \mathbf{u}\mathbf{G}$, where $\mathbf{u}$ is the k -bit long message and $\mathbf{G}$ is a k by n **generator matrix**
- The **parity check matrix** $\mathbf{H}$ is a $n-k$ by n matrix of ones and zeros, such that if $\mathbf{c}$ is a valid codeword then, $\mathbf{c}\mathbf{H}^T = \mathbf{0}$
- Each row of $\mathbf{H}$ specifies a parity check equation. The code bits in positions where the row is one must sum (modulo-2) to zero

6/7/2006

Turbo and LDPC Codes

68/133

Low-Density Parity-Check Codes

- **Low-Density Parity-Check** (LDPC) codes are a class of linear block codes characterized by sparse parity check matrices $\mathbf{H}$
 - $\mathbf{H}$ has a low-density of 1's
- LDPC codes were originally invented by Robert Gallager in the early 1960's but were largely ignored until they were "rediscovered" in the mid-1990's by MacKay
- Sparseness of $\mathbf{H}$ can yield large minimum distance d_{min} and reduces decoding complexity
- Can perform within 0.0045 dB of Shannon limit

6/7/2006

Turbo and LDPC Codes

69/133

Decoding LDPC codes

- Like Turbo codes, LDPC can be decoded iteratively
 - Instead of a trellis, the decoding takes place on a **Tanner graph**
 - Messages are exchanged between the v-nodes and c-nodes
 - Edges of the graph act as *information pathways*
- Hard decision decoding
 - *Bit-flipping* algorithm
- Soft decision decoding
 - *Sum-product* algorithm
 - Also known as message passing/ belief propagation algorithm
 - *Min-sum* algorithm
 - Reduced complexity approximation to the sum-product algorithm
- In general, the per-iteration complexity of LDPC codes is less than it is for turbo codes
 - However, many more iterations may be required (max $\approx$ 100; avg $\approx$ 30)
 - Thus, *overall* complexity can be higher than turbo

6/7/2006

Turbo and LDPC Codes

70/133

Tanner Graphs

- A **Tanner graph** is a *bipartite* graph that describes the parity check matrix **H**
- There are two classes of nodes:
 - **Variable-nodes**: Correspond to bits of the codeword or equivalently, to columns of the parity check matrix
 - There are n v-nodes
 - **Check-nodes**: Correspond to parity check equations or equivalently, to rows of the parity check matrix
 - There are $m=n-k$ c-nodes
 - **Bipartite** means that nodes of the same type cannot be connected (e.g. a c-node cannot be connected to another c-node)
- The i^{th} check node is connected to the j^{th} variable node iff the $(i,j)^{\text{th}}$ element of the parity check matrix is one, i.e. if $h_{ij}=1$
 - All of the v-nodes connected to a particular c-node must sum (modulo-2) to zero

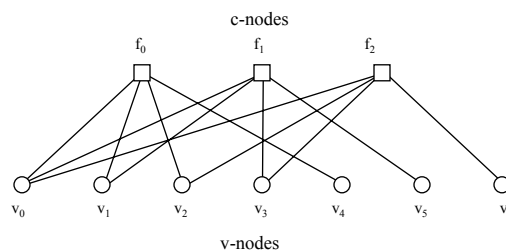
6/7/2006

Turbo and LDPC Codes

71/133

Example: Tanner Graph for (7,4) Hamming Code

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



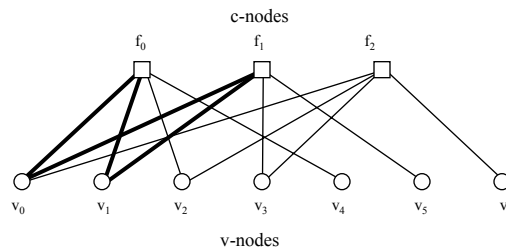
6/7/2006

Turbo and LDPC Codes

72/133

More on Tanner Graphs

- A **cycle** of length l in a Tanner graph is a path of l distinct edges which closes on itself
- The **girth** of a Tanner graph is the minimum cycle length of the graph.
 - The shortest possible cycle in a Tanner graph has length 4

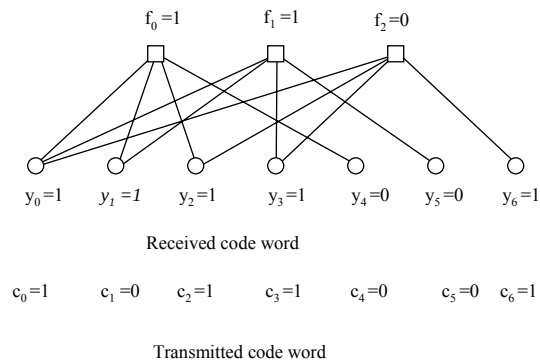


6/7/2006

Turbo and LDPC Codes

73/133

Bit-Flipping Algorithm: (7,4) Hamming Code

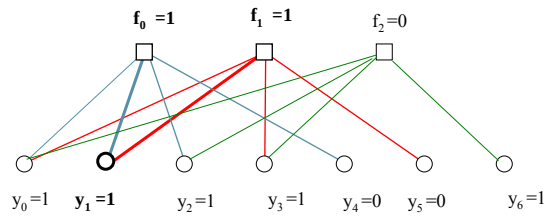


6/7/2006

Turbo and LDPC Codes

74/133

Bit-Flipping Algorithm: (7,4) Hamming Code

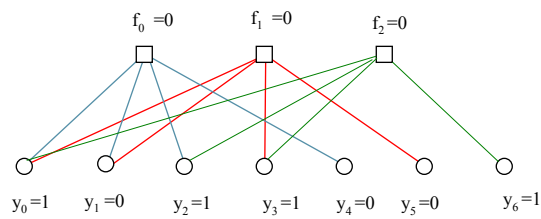


6/7/2006

Turbo and LDPC Codes

75/133

Bit-Flipping Algorithm: (7,4) Hamming Code



6/7/2006

Turbo and LDPC Codes

76/133

Generalized Bit-Flipping Algorithm

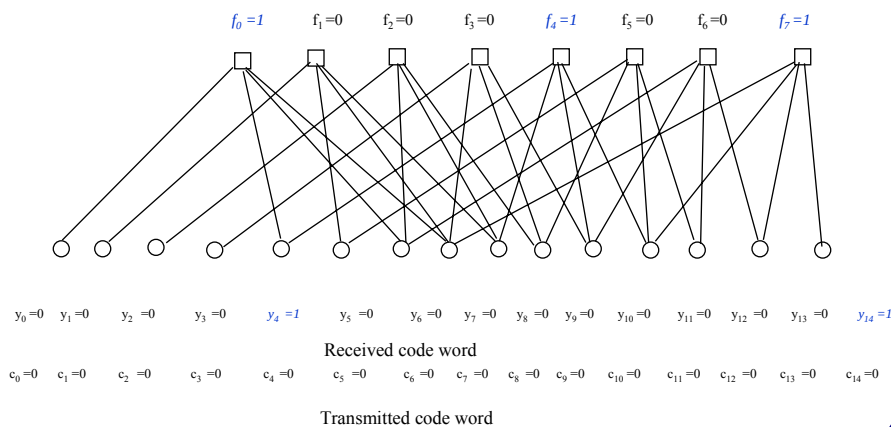
- Step 1: Compute parity-checks
 - If all checks are zero, stop decoding
- Step 2: Flip any digit contained in T or more failed check equations
- Step 3: Repeat 1 to 2 until all the parity checks are zero or a maximum number of iterations are reached
- The parameter T can be varied for a faster convergence

6/7/2006

Turbo and LDPC Codes

77/133

Generalized Bit Flipping: (15,7) BCH Code

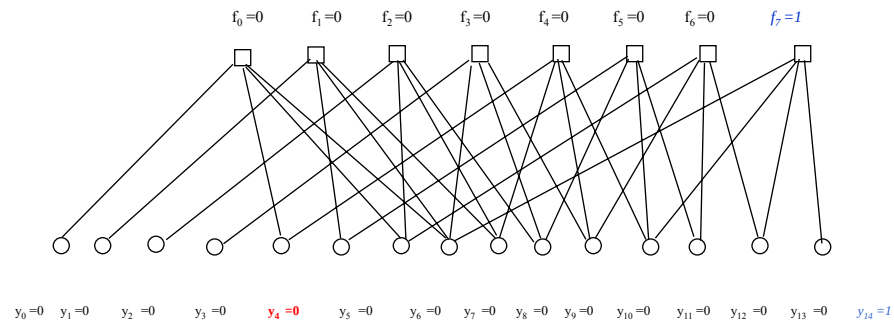


6/7/2006

Turbo and LDPC Codes

78/133

Generalized Bit Flipping: (15,7) BCH Code

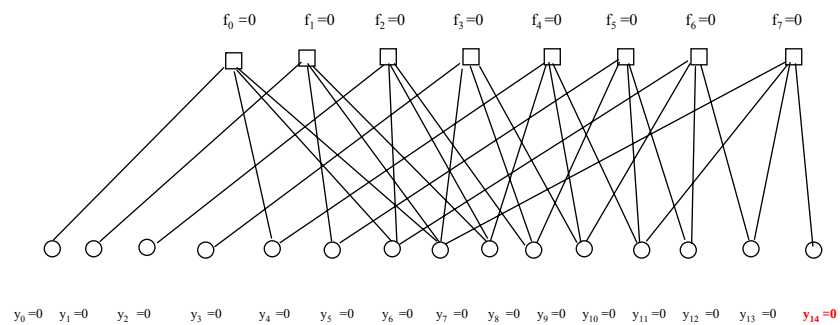


6/7/2006

Turbo and LDPC Codes

79/133

Generalized Bit Flipping: (15,7) BCH Code



6/7/2006

Turbo and LDPC Codes

80/133

Sum-Product Algorithm: Notation

- $Q_0 = P(c_i = 0 | y, S_i)$, $Q_1 = P(c_i = 1 | y, S_i)$
- S_i = event that bits in c satisfy the d_v parity check equations involving c_i
- $q_{ij}(b)$ = extrinsic info to be passed from v-node i to c-node j
 - Probability that $c_i = b$ given extrinsic information from check nodes and channel sample y_i
- $r_{ji}(b)$ = extrinsic info to be passed from c-node j to v-node i
 - Probability of the j^{th} check equation being satisfied given that $c_i = b$
- $C_i = \{j: h_{ji} = 1\}$
 - This is the set of row location of the 1's in the i^{th} column
- $C_{i\setminus j} = \{j': h_{ji'} = 1\} \setminus \{j\}$
 - The set of row locations of the 1's in the i^{th} column, excluding location j
- $R_j = \{i: h_{ji} = 1\}$
 - This is the set of column location of the 1's in the j^{th} row
- $R_{j\setminus i} = \{i': h_{ji'} = 1\} \setminus \{i\}$
 - The set of column locations of the 1's in the j^{th} row, excluding location i

6/7/2006

Turbo and LDPC Codes

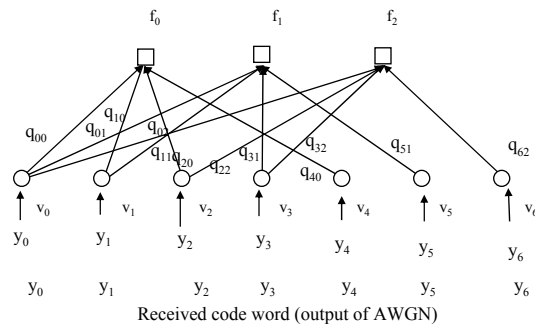
81/133

Sum-Product Algorithm

Step 1: Initialize $q_{ij}(0) = 1 - p_i = 1 / (1 + \exp(-2y_i / \sigma^2))$

$q_{ij}(1) = p_i = 1 / (1 + \exp(2y_i / \sigma^2))$

$q_{ij}(b)$ = probability that $c_i = b$, given the channel sample



6/7/2006

Turbo and LDPC Codes

82/133

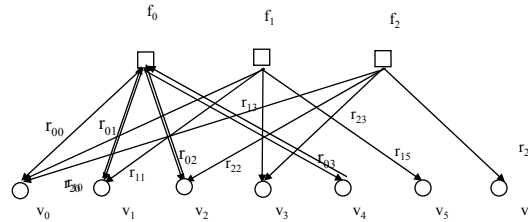
Sum-Product Algorithm

Step 2: At each c-node, update the r messages

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{l \in R_{ji}} (1 - 2q_{lj}(1))$$

$$r_{ji}(1) = 1 - r_{ji}(0)$$

$r_{ji}(b)$ = probability that j^{th} check equation is satisfied given $c_i = b$



6/7/2006

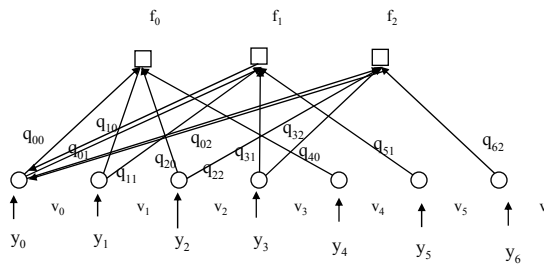
Turbo and LDPC Codes

83/133

Sum-Product Algorithm

Step 3: Update $q_{ij}(0)$ and $q_{ij}(1)$

$$q_{ij}(0) = k_{ij}(1 - p_i) \prod_{j' \in C_{ij}} (r_{j'}(0))$$

$$q_{ij}(1) = k_{ij}(p_i) \prod_{j' \in C_{ij}} (r_{j'}(1))$$


$Q_i(0) = k_{ij}(1 - p_i) \prod_{j \in C_i} (r_j(0))$

$Q_i(1) = k_{ij}(p_i) \prod_{j \in C_i} (r_j(1))$

Make hard decision

$$\hat{c}_i = \begin{cases} 1 & \text{if } Q_i(0) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

6/7/2006

Turbo and LDPC Codes

84/133

Halting Criteria

- After each iteration, halt if:

$$\hat{\mathbf{c}}\mathbf{H}^T = \mathbf{0}$$

- This is effective, because the probability of an undetectable decoding error is negligible
- Otherwise, halt once the maximum number of iterations is reached
- If the Tanner graph contains no cycles, then Q_i converges to the true APP value as the number of iterations tends to infinity

6/7/2006

Turbo and LDPC Codes

85/133

Sum-Product Algorithm in Log Domain

- The sum-product algorithm in probability domain has two shortcomings
 - Numerically unstable
 - Too many multiplications
- A log domain version is often used for practical purposes
- $Q_i = \log \left(\frac{P(c_i = 0 | \underline{y}, S_i)}{P(c_i = 1 | \underline{y}, S_i)} \right)$ LLR of the i^{th} code bit (ultimate goal of algorithm)
- $q_{ij} = \log (q_{ij}(0)/q_{ij}(1))$ extrinsic info to be passed from v-node i to c-node j
- $r_{ji} = \log (r_{ji}(0)/r_{ji}(1))$ extrinsic info to be passed from c-node j to v-node i

6/7/2006

Turbo and LDPC Codes

86/133

Sum-Product Decoder (in Log-Domain)

- Initialize:
 - $q_{ij} = \lambda_i = 2y_i/\sigma^2 = \text{channel LLR value}$
- Loop over all i, j for which $h_{ij} = 1$
 - At each c-node, update the r messages:

$$r_{ji} = \left(\prod_{i' \in R_{ji}} \alpha_{i'j} \right) \phi \left(\sum_{i' \in R_{ji}} \phi(\beta_{i'j}) \right)$$

- At each v-node update the q message and Q LLR:

$$Q_i = \lambda_i + \sum_{j \in C_i} r_{ji}$$

$$q_{ij} = Q_i - r_{ji}$$

- Make hard decision:

$$\hat{c}_i = \begin{cases} 1 & \text{if } Q_i < 0 \\ 0 & \text{otherwise} \end{cases}$$

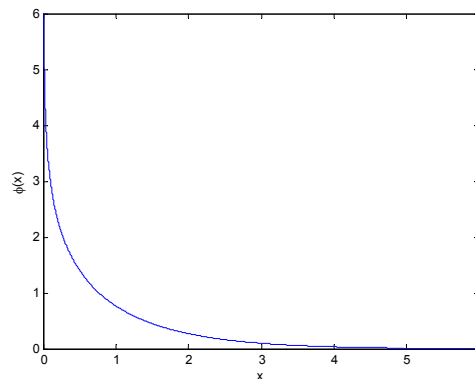
6/7/2006

Turbo and LDPC Codes

87/133

Sum-Product Algorithm: Notation

- $\alpha_{ij} = \text{sign}(q_{ij})$
- $\beta_{ij} = |q_{ij}|$
- $\phi(x) = -\log \tanh(x/2) = \log((e^x+1)/(e^x-1)) = \phi^{-1}(x)$



6/7/2006

Turbo and LDPC Codes

88/133

Min-Sum Algorithm

- Note that:

$$\phi\left(\sum_i \phi(\beta_{i,j})\right) \approx \phi\left(\phi\left(\min_i \beta_{i,j}\right)\right) = \min_i \beta_{i,j}$$

- So we can replace the r message update formula with

$$r_{ji} = \left(\prod_{i' \in R_{ji}} \alpha_{i'j} \right) \min_{i' \in R_{ji}} \beta_{i'j}$$

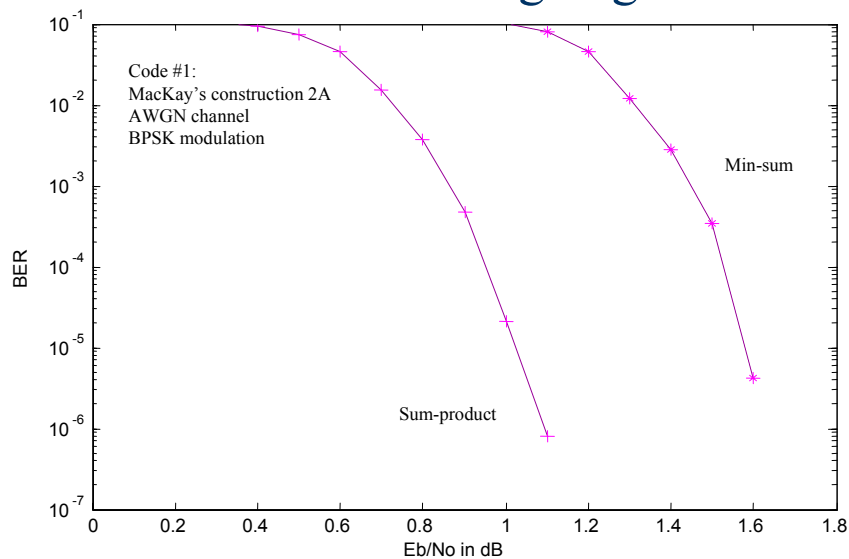
- This greatly reduces complexity, since now we don't have to worry about computing the nonlinear ϕ function.
- Note that since α is just the sign of q , $\prod \alpha$ can be implemented by using XOR operations.

6/7/2006

Turbo and LDPC Codes

89/133

BER of Different Decoding Algorithms



Extrinsic-information Scaling

- As with max-log-MAP decoding of turbo codes, min-sum decoding of LDPC codes produces an extrinsic information estimate which is biased.
 - In particular, r_{ji} is overly optimistic.
- A significant performance improvement can be achieved by multiplying r_{ji} by a constant κ , where $\kappa < 1$.

$$r_{ji} = \kappa \left(\prod_{i' \in R_{ji}} \alpha_{i'j} \right) \min_{i' \in R_{ji}} \beta_{i'j}$$

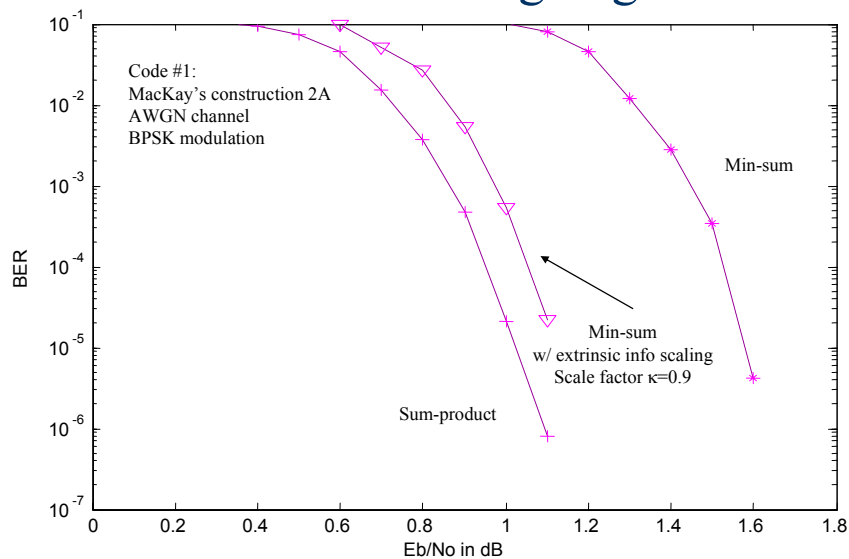
- See: J. Heo, "Analysis of scaling soft information on low density parity check code," *IEEE Electronic Letters*, 23rd Jan. 2003.
- Experimentation shows that $\kappa=0.9$ gives best performance.

6/7/2006

Turbo and LDPC Codes

91/133

BER of Different Decoding Algorithms



Regular vs. Irregular LDPC codes

- An LDPC code is **regular** if the rows and columns of H have uniform weight, i.e. all rows have the same number of ones (d_v) and all columns have the same number of ones (d_c)
 - The codes of Gallager and MacKay were regular (or as close as possible)
 - Although regular codes had impressive performance, they are still about 1 dB from capacity and generally perform worse than turbo codes
- An LDPC code is **irregular** if the rows and columns have non-uniform weight
 - Irregular LDPC codes tend to outperform turbo codes for block lengths of about $n > 10^5$
- The degree distribution pair (λ, ρ) for a LDPC code is defined as

$$\lambda(x) = \sum_{i=2}^{d_v} \lambda_i x^{i-1}$$

$$\rho(x) = \sum_{i=1}^{d_c} \rho_i x^{i-1}$$
- λ_i, ρ_i represent the fraction of edges emanating from variable (check) nodes of degree i

6/7/2006

Turbo and LDPC Codes

93/133

Constructing Regular LDPC Codes: MacKay, 1996

- Around 1996, Mackay and Neal described methods for constructing sparse H matrices
- The idea is to randomly generate a $M \times N$ matrix H with weight d_v columns and weight d_c rows, subject to some constraints
- Construction 1A: Overlap between any two columns is no greater than 1
 - This avoids length 4 cycles
- Construction 2A: $M/2$ columns have $d_v = 2$, with no overlap between any pair of columns. Remaining columns have $d_v = 3$. As with 1A, the overlap between any two columns is no greater than 1
- Construction 1B and 2B: Obtained by deleting select columns from 1A and 2A
 - Can result in a higher rate code

6/7/2006

Turbo and LDPC Codes

94/133

Constructing Irregular LDPC Codes: Luby, et. al., 1998

- Luby et. al. (1998) developed LDPC codes based on irregular LDPC Tanner graphs
- Message and check nodes have conflicting requirements
 - Message nodes benefit from having a large degree
 - LDPC codes perform better with check nodes having low degrees
- Irregular LDPC codes help balance these competing requirements
 - High degree message nodes converge to the correct value quickly
 - This increases the quality of information passed to the check nodes, which in turn helps the lower degree message nodes to converge
- Check node degree kept as uniform as possible and variable node degree is non-uniform
 - Code 14: Check node degree =14, Variable node degree =5, 6, 21, 23
- No attempt made to optimize the degree distribution for a given code rate

6/7/2006

Turbo and LDPC Codes

95/133

Density Evolution: Richardson and Urbanke, 2001

- Given an irregular Tanner graph with a maximum d_v and d_c , what is the best degree distribution?
 - How many of the v-nodes should be degree $d_v, d_{v-1}, d_{v-2}, \dots$ nodes?
 - How many of the c-nodes should be degree $d_c, d_{c-1}, \dots$ nodes?
- Question answered using **Density Evolution**
 - Process of tracking the evolution of the message distribution during belief propagation
- For any LDPC code, there is a “worst case” channel parameter called the *threshold* such that the message distribution during belief propagation evolves in such a way that the probability of error converges to zero as the number of iterations tends to infinity
- Density evolution is used to find the degree distribution pair (λ, ρ) that maximizes this threshold

6/7/2006

Turbo and LDPC Codes

96/133

Density Evolution: Richardson and Urbanke, 2001

- Step 1: Fix a maximum number of iterations
- Step 2: For an initial degree distribution, find the threshold
- Step 3: Apply a small change to the degree distribution
 - If the new threshold is larger, fix this as the current distribution
- Repeat Steps 2-3
- Richardson and Urbanke identify a rate $\frac{1}{2}$ code with degree distribution pair which is 0.06 dB away from capacity
 - “Design of capacity-approaching irregular low-density parity-check codes”, *IEEE Trans. Inf. Theory*, Feb. 2001
- Chung et.al., use density evolution to design a rate $\frac{1}{2}$ code which is 0.0045 dB away from capacity
 - “On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit”, *IEEE Comm. Letters*, Feb. 2001

6/7/2006

Turbo and LDPC Codes

97/133

More on Code Construction

- LDPC codes, especially irregular codes exhibit error floors at high SNRs
- The error floor is influenced by $d_{\min}$
 - Directly designing codes for large $d_{\min}$ is not computationally feasible
- Removing short cycles indirectly increases $d_{\min}$ (*girth conditioning*)
 - Not all short cycles cause error floors
- *Trapping sets* and *Stopping sets* have a more direct influence on the error floor
- Error floors can be mitigated by increasing the size of minimum stopping sets
 - Tian, et. al., “Construction of irregular LDPC codes with low error floors”, in *Proc. ICC*, 2003
- Trapping sets can be mitigated using *averaged belief propagation decoding*
 - Milenkovic, “Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes”, in *Proc. Intl. Conf. on Wireless Ntw., Communications and Mobile computing*, 2005
- LDPC codes based on *projective geometry* reported to have very low error floors
 - Kou, “Low-density parity-check codes based on finite geometries: a rediscovery and new results”, *IEEE Tans. Inf. Theory*, Nov. 1998

6/7/2006

Turbo and LDPC Codes

98/133

Encoding LDPC Codes

- A linear block code is encoded by performing the matrix multiplication $\mathbf{c} = \mathbf{uG}$
- A common method for finding $\mathbf{G}$ from $\mathbf{H}$ is to first make the code systematic by adding rows and exchanging columns to get the $\mathbf{H}$ matrix in the form $\mathbf{H} = [\mathbf{P}^T \mathbf{I}]$
 - Then $\mathbf{G} = [\mathbf{I} \mathbf{P}]$
 - However, the result of the row reduction is a non-sparse $\mathbf{P}$ matrix
 - The multiplication $\mathbf{c} = [\mathbf{u} \mathbf{uP}]$ is therefore very complex
- As an example, for a (10000, 5000) code, $\mathbf{P}$ is 5000 by 5000
 - Assuming the density of 1's in $\mathbf{P}$ is 0.5, then $0.5 \times (5000)^2$ additions are required per codeword
- This is especially problematic since we are interested in large n ($>10^5$)
- An often used approach is to use the all-zero codeword in simulations

6/7/2006

Turbo and LDPC Codes

99/133

Encoding LDPC Codes

- Richardson and Urbanke show that even for large n , the encoding complexity can be (almost) linear function of n
 - “Efficient encoding of low-density parity-check codes”, *IEEE Trans. Inf. Theory*, Feb., 2001
- Using only row and column permutations, $\mathbf{H}$ is converted to an approximately lower triangular matrix
 - Since only permutations are used, $\mathbf{H}$ is still sparse
 - The resulting encoding complexity is almost linear as a function of n
- An alternative involving a sparse-matrix multiply followed by differential encoding has been proposed by Ryan, Yang, & Li...
 - “Lowering the error-rate floors of moderate-length high-rate irregular LDPC codes,” *ISIT*, 2003

6/7/2006

Turbo and LDPC Codes

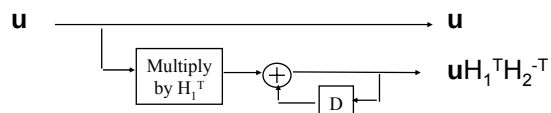
100/133

Encoding LDPC Codes

- Let $H = [H_1 \ H_2]$ where H_1 is sparse and

$$H_2 = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \dots & 1 \\ & & & & 1 & 1 \end{bmatrix} \quad \text{and} \quad H_2^{-T} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ & 1 & 1 & \dots & 1 \\ & & 1 & \dots & 1 \\ & & & \dots & 1 \\ & & & & 1 \end{bmatrix}$$

- Then a systematic code can be generated with $G = [I \ H_1^T H_2^{-T}]$.
- It turns out that H_2^{-T} is the generator matrix for an accumulate-code (differential encoder), and thus the encoder structure is simply:



- Similar to Jin & McEliece's Irregular Repeat Accumulate (IRA) codes.
 - Thus termed **"Extended IRA Codes"**

6/7/2006

Turbo and LDPC Codes

101/133

Performance Comparison

- We now compare the performance of the maximum-length UMTS turbo code against four LDPC code designs.
- Code parameters
 - All codes are rate $\frac{1}{3}$
 - The LDPC codes are length $(n,k) = (15000, 5000)$
 - Up to 100 iterations of log-domain sum-product decoding
 - Code parameters are given on next slide
 - The turbo code has length $(n,k) = (15354, 5114)$
 - Up to 16 iterations of log-MAP decoding
- BPSK modulation
- AWGN and fully-interleaved Rayleigh fading
- Enough trials run to log 40 frame errors
 - Sometimes fewer trials were run for the last point (highest SNR).

6/7/2006

Turbo and LDPC Codes

102/133

LDPC Code Parameters

- Code 1: MacKay's regular construction 2A
 - See: D.J.C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, March 1999.
- Code 2: Richardson & Urbanke irregular construction
 - See T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, Feb. 2001.
- Code 3: Improved irregular construction
 - Designed by Chris Jones using principles from T. Tian, C. Jones, J.D. Villasenor, and R.D. Wesel, "Construction of irregular LDPC codes with low error floors," in *Proc. ICC 2003*.
 - Idea is to avoid small stopping sets
- Code 4: Extended IRA code
 - Designed by Michael Yang & Bill Ryan using principles from M. Yang and W.E. Ryan, "Lowering the error-rate floors of moderate-length high-rate irregular LDPC codes," *ISIT*, 2003.

6/7/2006

Turbo and LDPC Codes

103/133

LDPC Degree Distributions

- The distribution of row-weights, or check-node degrees, is as follows:

i	1	2	3	4
3				1
4	10000			4999
5		13	5458	5000
6		9987	4542	

Code number:
 1 = MacKay construction 2A
 2 = Richardson & Urbanke
 3 = Jones, Wesel, & Tian
 4 = Ryan's Extended-IRA

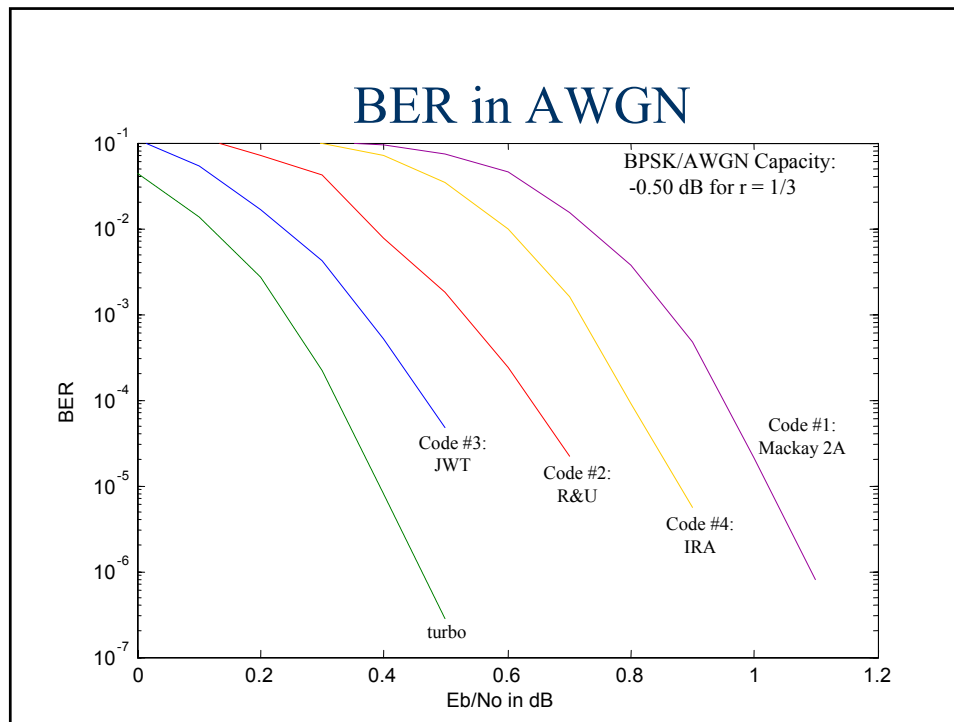
- The distribution of column-weights, or variable-node degrees, is:

i	1	2	3	4
1				1
2	5000	8282	9045	9999
3	10000	2238	2267	
4		2584	569	5000
5		206	1941	
8		1		
15		1689	1178	

6/7/2006

Turbo and LDPC Codes

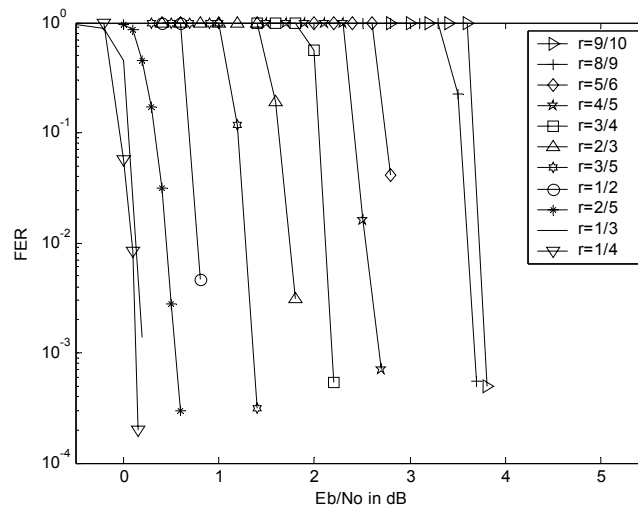
104/133



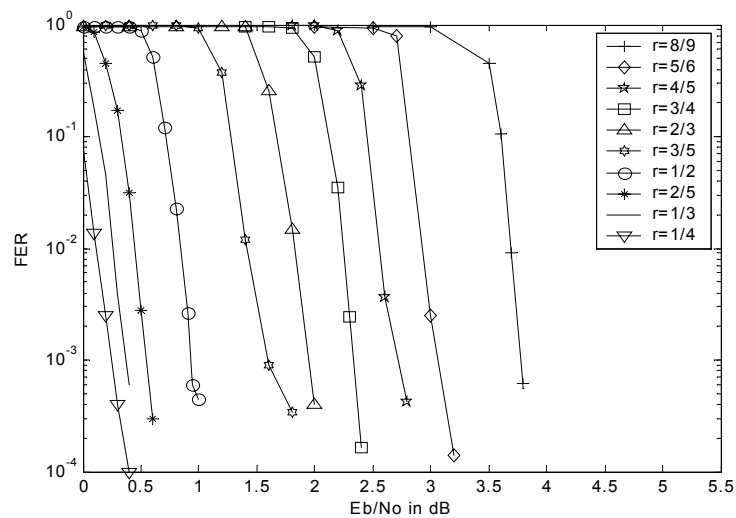
DVB-S2 LDPC Code

- The digital video broadcasting (DVB) project was founded in 1993 by ETSI to standardize digital television services
- The latest version of the standard DVB-S2 uses a concatenation of an outer BCH code and inner LDPC code
- The codeword length can be either $n = 64800$ (normal frames) or $n = 16200$ (short frames)
- Normal frames support code rates $9/10, 8/9, 5/6, 4/5, 3/4, 2/3, 3/5, 1/2, 2/5, 1/3, 1/4$
 - Short frames do not support rate $9/10$
- DVB-S2 uses an extended-IRA type LDPC code
- Valenti, et. al, "Turbo and LDPC codes for digital video broadcasting," Chapter 12 of *Turbo Code Application: A Journey from a Paper to Realizations*, Springer, 2005.

FER for DVB-S2 LDPC Code Normal Frames in BPSK/AWGN



FER for DVB-S2 LDPC Code Short Frames in BPSK/AWGN



M-ary Complex Modulation

- $\mu = \log_2 M$ bits are mapped to the symbol $\mathbf{x}_k$, which is chosen from the set $S = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M\}$
 - The symbol is multidimensional.
 - 2-D Examples: QPSK, M-PSK, QAM, APSK, HEX
 - M-D Example: FSK, block space-time codes (BSTC)
- The signal $\mathbf{y} = \mathbf{h}\mathbf{x}_k + \mathbf{n}$ is received
 - $\mathbf{h}$ is a complex fading coefficient.
 - More generally (BSTC), $\mathbf{Y} = \mathbf{H}\mathbf{X} + \mathbf{N}$
- Modulation implementation in the ISCMML
 - The complex signal set S is created with the **CreateConstellation** function.
 - Modulation is performed using the **Modulate** function.

6/7/2006

Turbo and LDPC Codes

109/133

Log-likelihood of Received Symbols

- Let $p(\mathbf{x}_k|\mathbf{y})$ denote the probability that signal $\mathbf{x}_k \in S$ was transmitted given that $\mathbf{y}$ was received.
- Let $f(\mathbf{x}_k|\mathbf{y}) = K p(\mathbf{x}_k|\mathbf{y})$, where K is any multiplicative term that is constant for all $\mathbf{x}_k$.
- When all symbols are equally likely, $f(\mathbf{x}_k|\mathbf{y}) \propto f(\mathbf{y}|\mathbf{x}_k)$
- For each signal in S , the receiver computes $f(\mathbf{y}|\mathbf{x}_k)$
 - This function depends on the modulation, channel, and receiver.
 - Implemented by the **Demod2D** and **DemodFSK** functions, which actually computes $\log f(\mathbf{y}|\mathbf{x}_k)$.
- Assuming that all symbols are equally likely, the most likely symbol $\mathbf{x}_k$ is found by making a hard decision on $f(\mathbf{y}|\mathbf{x}_k)$ or $\log f(\mathbf{y}|\mathbf{x}_k)$.

6/7/2006

Turbo and LDPC Codes

110/133

Example: QAM over AWGN.

- Let $y = x + n$, where n is complex i.i.d. $N(0, N_0/2)$ and the average energy per symbol is $E[|x|^2] = E_s$

$$p(y|x_k) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{|y-x_k|^2}{2\sigma^2}\right\}$$

$$f(y|x_k) = \exp\left\{-\frac{|y-x_k|^2}{2\sigma^2}\right\}$$

$$\begin{aligned} \log f(y|x_k) &= \frac{-|y-x_k|^2}{2\sigma^2} \\ &= \frac{-E_s|y-x_k|^2}{N_0} \end{aligned}$$

6/7/2006

Turbo and LDPC Codes

111/133

Log-Likelihood of Symbol x_k

- The log-likelihood of symbol x_k is found by:

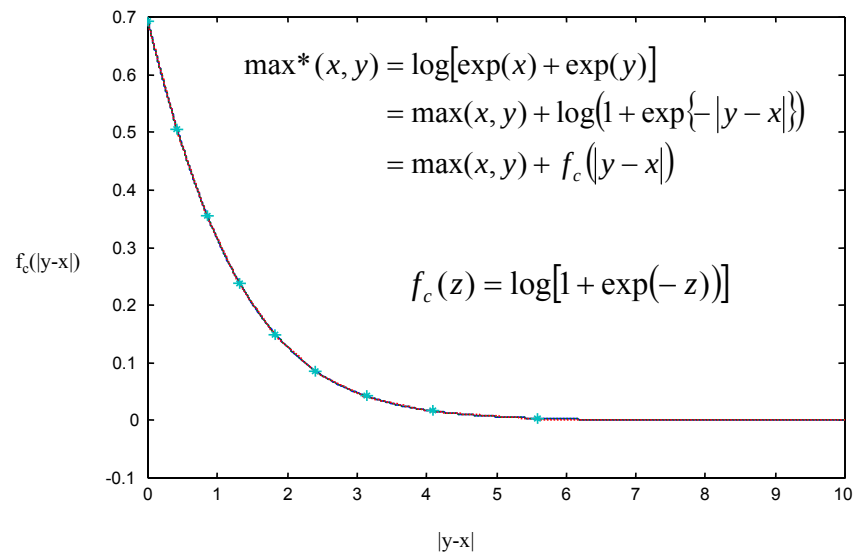
$$\begin{aligned} \Lambda_k &= \log p(\mathbf{x}_k | \mathbf{y}) \\ &= \log \frac{p(\mathbf{x}_k | \mathbf{y})}{\sum_{\mathbf{x}_m \in S} p(\mathbf{x}_m | \mathbf{y})} \\ &= \log \frac{f(\mathbf{y} | \mathbf{x}_k)}{\sum_{\mathbf{x}_m \in S} f(\mathbf{y} | \mathbf{x}_m)} \\ &= \log f(\mathbf{y} | \mathbf{x}_k) - \log \sum_{\mathbf{x}_m \in S} f(\mathbf{y} | \mathbf{x}_m) \\ &= \log f(\mathbf{y} | \mathbf{x}_k) - \log \sum_{\mathbf{x}_m \in S} \exp\{\log f(\mathbf{y} | \mathbf{x}_m)\} \\ &= \log f(\mathbf{y} | \mathbf{x}_k) - \max_{\mathbf{x}_m \in S} [\log f(\mathbf{y} | \mathbf{x}_m)] \end{aligned}$$

6/7/2006

Turbo and LDPC Codes

112/133

The max* function



Capacity of Coded Modulation (CM)

- Suppose we want to compute capacity of M-ary modulation
 - In each case, the input distribution is constrained, so there is no need to maximize over $p(x)$
 - The capacity is merely the mutual information between channel input and output.
- The mutual information can be measured as the following expectation:

$$C = I(X; Y) = E_{\mathbf{x}_k, \mathbf{y}} [\log M + \log p(\mathbf{x}_k | \mathbf{y})] \text{ nats}$$

Monte Carlo Calculation of the Capacity of Coded Modulation (CM)

- The mutual information can be measured as the following expectation:

$$\begin{aligned}
 C &= I(X; Y) = E_{\mathbf{x}_k, \mathbf{n}} [\log M + \log p(\mathbf{x}_k | \mathbf{y})] \text{ nats} \\
 &= \log M + E_{\mathbf{x}_k, \mathbf{n}} [\Lambda_k] \text{ nats} \\
 &= \log_2 M + \frac{E_{\mathbf{x}_k, \mathbf{n}} [\Lambda_k]}{\log(2)} \text{ bits} \\
 &= \mu + \frac{E_{\mathbf{x}_k, \mathbf{n}} [\Lambda_k]}{\log(2)} \text{ bits}
 \end{aligned}$$

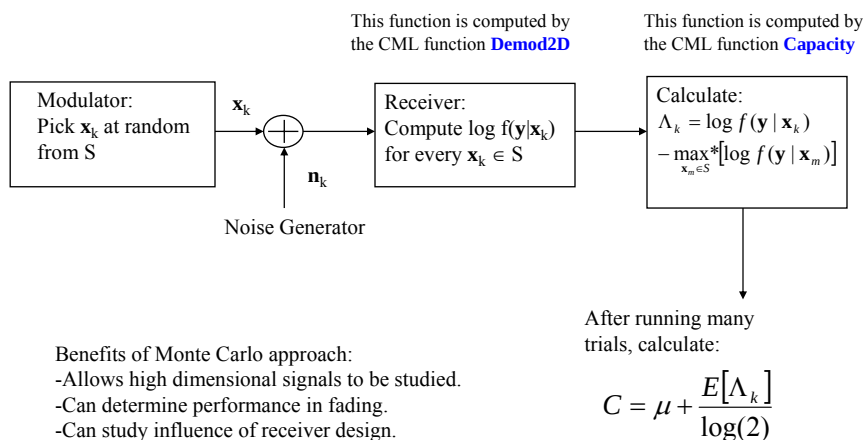
- This expectation can be obtained through Monte Carlo simulation.

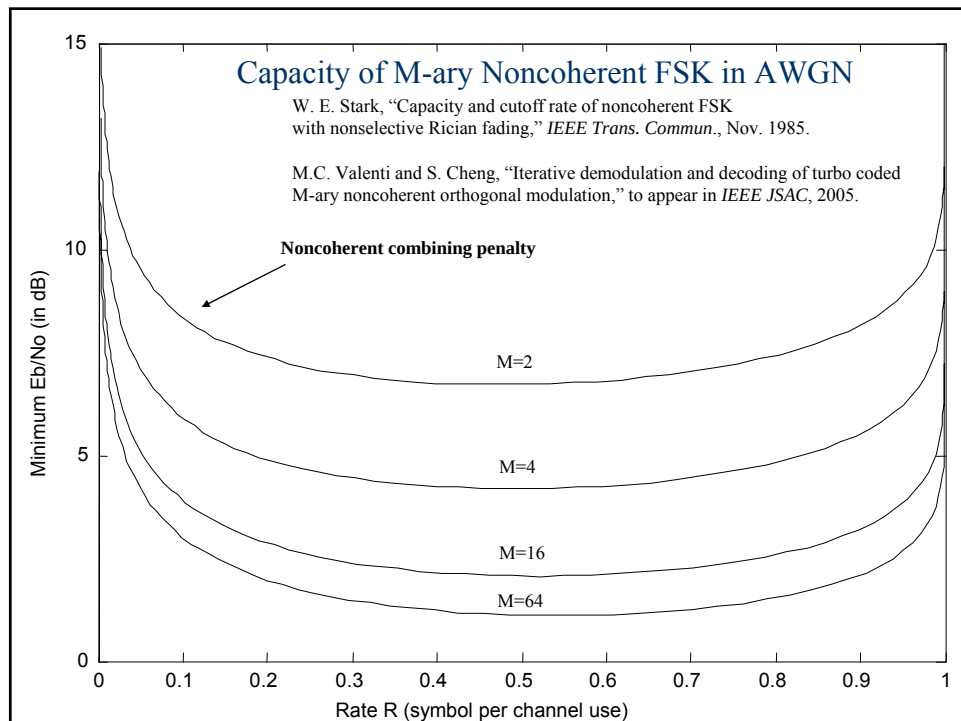
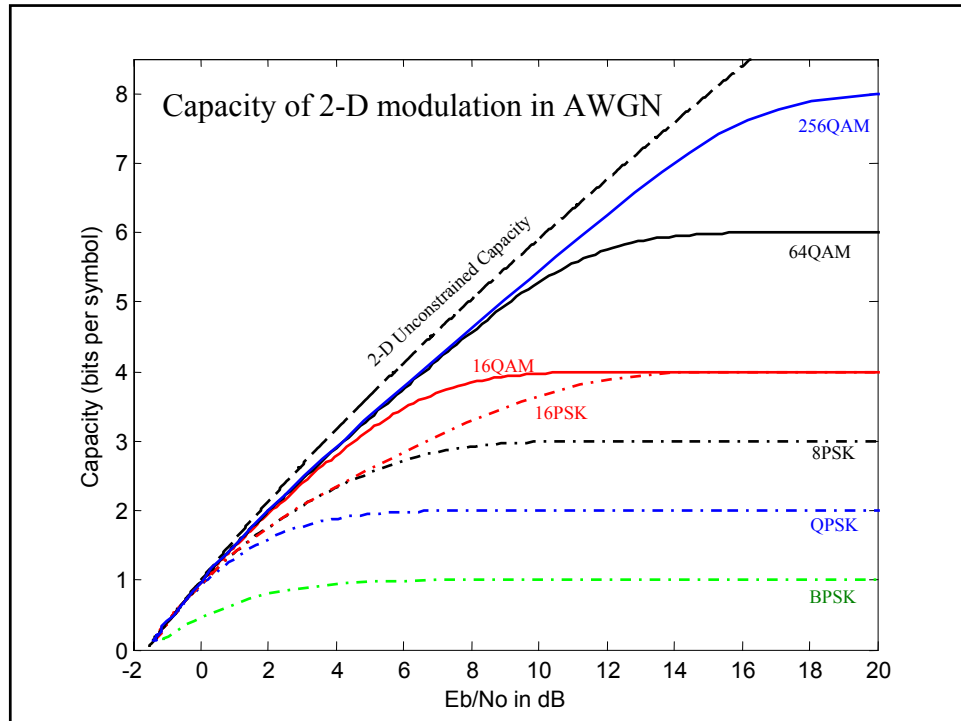
6/7/2006

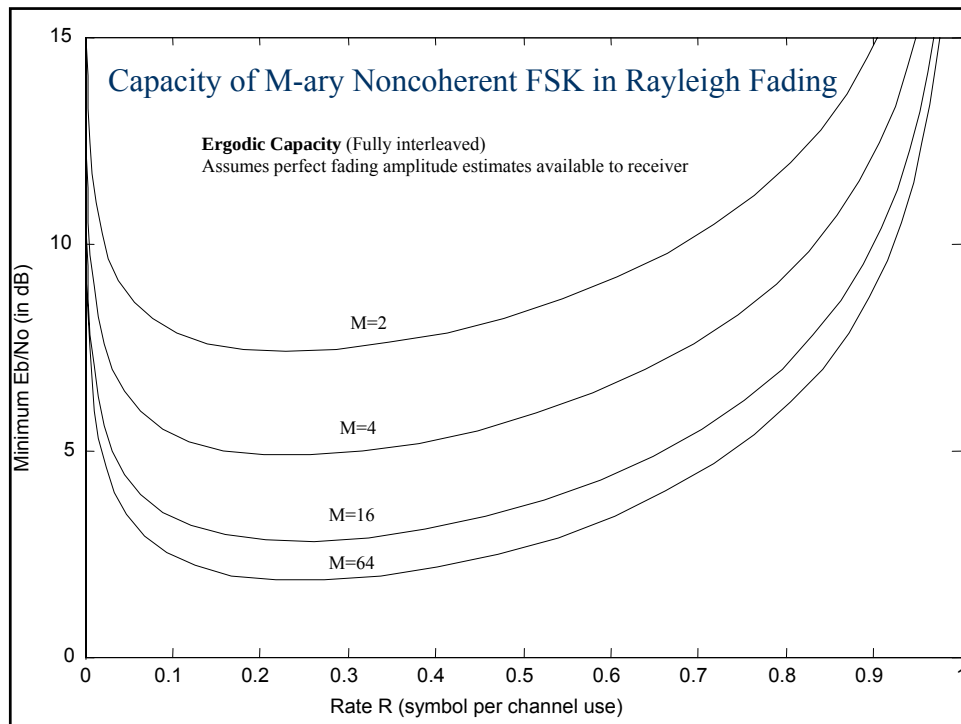
Turbo and LDPC Codes

115/133

Simulation Block Diagram







BICM

- Coded modulation (CM) is required to attain the aforementioned capacity.
 - Channel coding and modulation handled jointly.
 - e.g. trellis coded modulation (Ungerboeck); coset codes (Forney)
- Most off-the-shelf capacity approaching codes are binary.
- A pragmatic system would use a binary code followed by a bitwise interleaver and an M-ary modulator.
 - Bit Interleaved Coded Modulation (BICM); Caire 1998.



Transforming Symbol Log-Likehoods Into Bit LLRs

- Like the CM receiver, the BICM receiver calculates $\log f(\mathbf{y}|\mathbf{x}_k)$ for each signal in S .
- Furthermore, the BICM receiver needs to calculate the log-likelihood ratio of each code bit:

$$\lambda_n = \log \frac{p(c_n = 1 | \mathbf{y})}{p(c_n = 0 | \mathbf{y})} = \log \frac{\sum_{\mathbf{x}_k \in S_n^{(1)}} p(\mathbf{x}_k | \mathbf{y})}{\sum_{\mathbf{x}_k \in S_n^{(0)}} p(\mathbf{x}_k | \mathbf{y})} = \log \frac{\sum_{\mathbf{x}_k \in S_n^{(1)}} p(\mathbf{y} | \mathbf{x}_k) p[\mathbf{x}_k]}{\sum_{\mathbf{x}_k \in S_n^{(0)}} p(\mathbf{y} | \mathbf{x}_k) p[\mathbf{x}_k]}$$

$$= \max_{\mathbf{x}_k \in S_n^{(1)}}^* \log[f(\mathbf{y}|\mathbf{x}_k)] - \max_{\mathbf{x}_k \in S_n^{(0)}}^* \log[f(\mathbf{y}|\mathbf{x}_k)]$$

- where $S_n^{(1)}$ represents the set of symbols whose n^{th} bit is a 1.
- and $S_n^{(0)}$ is the set of symbols whose n^{th} bit is a 0.

6/7/2006

Turbo and LDPC Codes

121/133

BICM Capacity

- BICM transforms the channel into μ parallel binary channels, and the capacity of the n^{th} channel is:

$$\begin{aligned} C_k &= E_{c_n, n} [\log(2) + \log p(c_k | \mathbf{y})] \text{ nats} \\ &= \log(2) + E_{c_n, n} \left[\log \frac{p(c_k | \mathbf{y})}{p(c_k = 0 | \mathbf{y}) + p(c_k = 1 | \mathbf{y})} \right] \text{ nats} \\ &= \log(2) - E_{c_n, n} \left[\log \frac{p(c_k = 0 | \mathbf{y}) + p(c_k = 1 | \mathbf{y})}{p(c_k | \mathbf{y})} \right] \text{ nats} \\ &= \log(2) - E_{c_n, n} \left[\log \left\{ \exp \log \frac{p(c_k = 0 | \mathbf{y})}{p(c_k | \mathbf{y})} + \exp \log \frac{p(c_k = 1 | \mathbf{y})}{p(c_k | \mathbf{y})} \right\} \right] \text{ nats} \\ &= \log(2) - E_{c_n, n} \left[\max^* \left\{ \log \frac{p(c_k = 0 | \mathbf{y})}{p(c_k | \mathbf{y})}, \log \frac{p(c_k = 1 | \mathbf{y})}{p(c_k | \mathbf{y})} \right\} \right] \text{ nats} \\ &= \log(2) - E_{c_n, n} [\max^* \{0, (-1)^{c_k} \lambda_k\}] \text{ nats} \end{aligned}$$

6/7/2006

Turbo and LDPC Codes

122/133

BICM Capacity (Continued)

- Since capacity over parallel channels adds, the capacity of BICM is:

$$\begin{aligned}
 C &= \sum_{k=1}^{\mu} C_k \\
 &= \sum_{k=1}^{\mu} \left\{ \log(2) - E_{c_n, n} \left[\max^* \left\{ 0, (-1)^{c_k} \lambda_k \right\} \right] \right\} \text{ nats} \\
 &= \mu \log(2) - \sum_{k=1}^{\mu} E_{c_n, n} \left[\max^* \left\{ 0, (-1)^{c_k} \lambda_k \right\} \right] \text{ nats} \\
 &= \mu - \frac{1}{\log(2)} \sum_{k=1}^{\mu} E_{c_n, n} \left[\max^* \left\{ 0, (-1)^{c_k} \lambda_k \right\} \right] \text{ bits}
 \end{aligned}$$

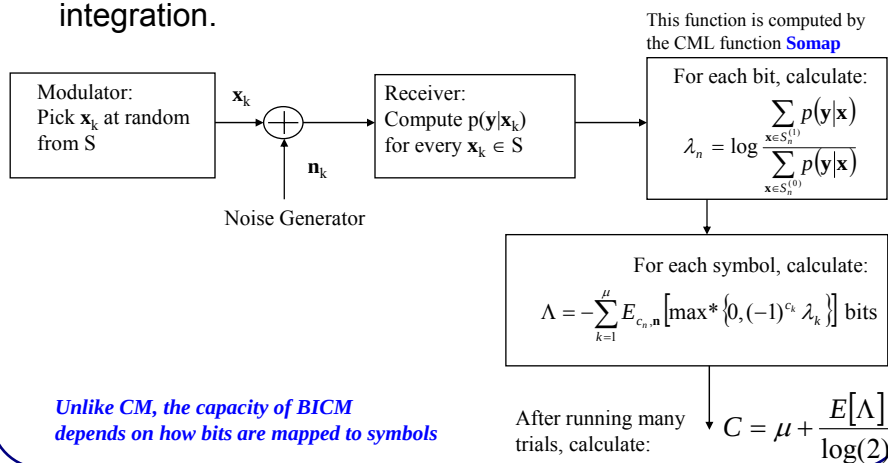
6/7/2006

Turbo and LDPC Codes

123/133

BICM Capacity

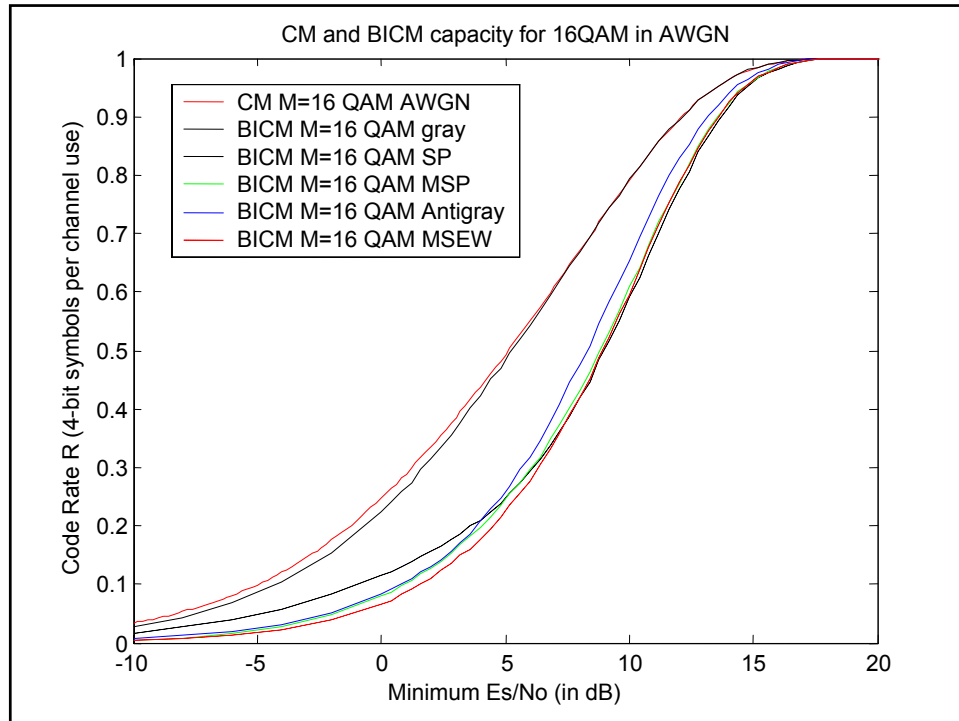
- As with CM, this can be computed using a Monte Carlo integration.



6/7/2006

Turbo and LDPC Codes

124/133



BICM-ID

- The conventional BICM receiver assumes that all bits in a symbol are equally likely:

$$\lambda_n = \log \frac{\sum_{\mathbf{x} \in S_n^{(1)}} p(\mathbf{x} | \mathbf{y})}{\sum_{\mathbf{x} \in S_n^{(0)}} p(\mathbf{x} | \mathbf{y})} = \log \frac{\sum_{\mathbf{x} \in S_n^{(1)}} p(\mathbf{y} | \mathbf{x})}{\sum_{\mathbf{x} \in S_n^{(0)}} p(\mathbf{y} | \mathbf{x})}$$

- However, if the receiver has estimates of the bit probabilities, it can use this to weight the symbol likelihoods.

$$\lambda_n = \log \frac{\sum_{\mathbf{x} \in S_n^{(1)}} p(\mathbf{y} | \mathbf{x}) p(\mathbf{x} | c_n = 1)}{\sum_{\mathbf{x} \in S_n^{(0)}} p(\mathbf{y} | \mathbf{x}) p(\mathbf{x} | c_n = 0)}$$

- This information is obtained from decoder feedback.
 - Bit Interleaved Coded Modulation with Iterative Demodulation
 - Li and Ritcey 1999.

Mutual Information Transfer Chart

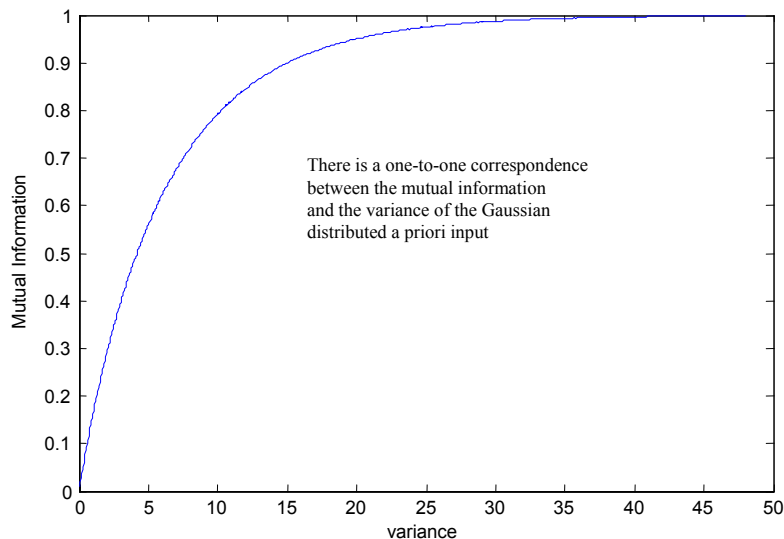
- Now consider a receiver that has a priori information about the code bits (from a soft output decoder).
- Assume the following:
 - The a priori information is in LLR form.
 - The a priori LLR's are Gaussian distributed.
 - The LLR's have mutual information I_v
- Then the mutual information I_z at the output of the receiver can be measured through Monte Carlo Integration.
 - I_z vs. I_v is the **Mutual Information Transfer Characteristic**.
 - ten Brink 1999.

6/7/2006

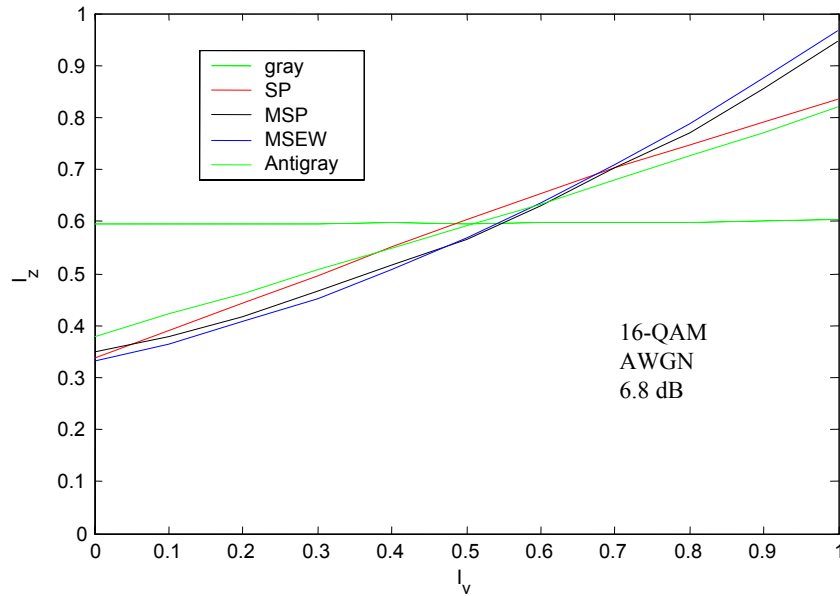
Turbo and LDPC Codes

127/133

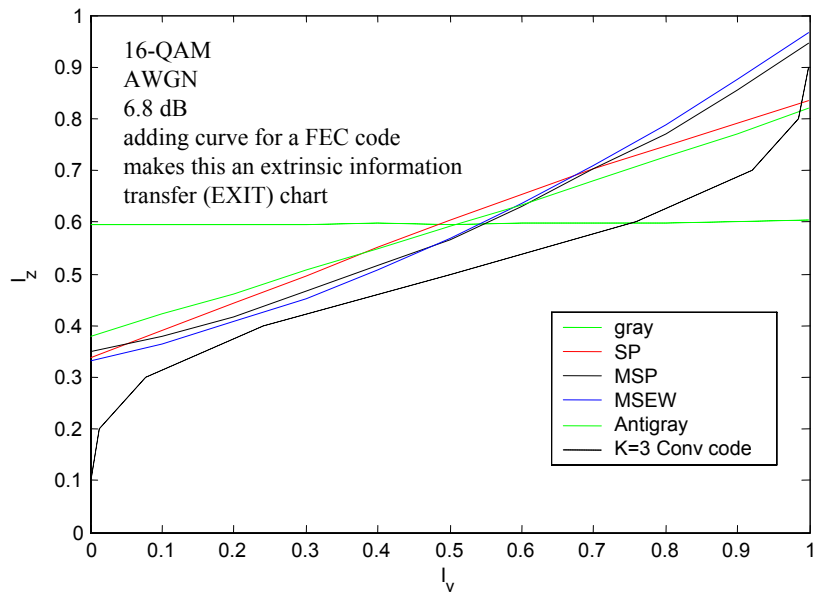
Generating Random a Priori Input



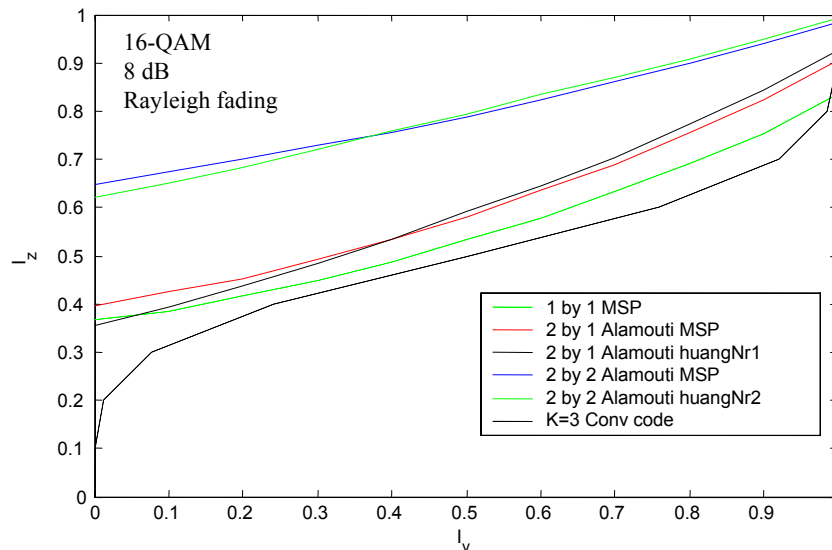
Mutual Information Characteristic



EXIT Chart



EXIT Chart for Space Time Block Code



EXIT Chart Analysis of Turbo Codes

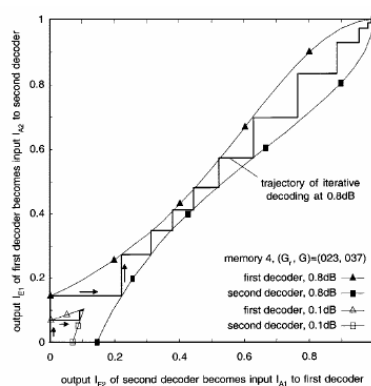


Fig. 5. Simulated trajectories of iterative decoding at $E_b/N_0 = 0.1$ dB and 0.8 dB (symmetric PCC rate 1/2, interleaver size 60 000 systematic bits).

- PCCC (turbo) codes can be analyzed with an EXIT chart by plotting the mutual information transfer characteristics of the two decoders.
- Figure is from:
S. ten Brink, "Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes," IEEE Trans. Commun., Oct. 2001.

Conclusions

- It is now possible to closely approach the Shannon limit by using turbo and LDPC codes.
- Binary capacity approaching codes can be combined with higher order modulation using the BICM principle.
- These code are making their way into standards
 - Binary turbo: UMTS, cdma2000
 - Duobinary turbo: DVB-RCS, 802.16
 - LDPC: DVB-S2 standard.
- Software for simulating turbo and LDPC codes can be found at www.iterativesolutions.com