Modern Coding Theory

Daniel J. Costello, Jr.

Coding Research Group
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

2009 School of Information Theory
Northwestern University
August 10, 2009

The author gratefully acknowledges the help of Ali Pusane and Christian Koller in the preparation of this presentation.
Outline

- Pre-1993 era (ancient coding theory)
  - Coding theory playing field
- Post-1993 era (modern coding theory)
  - Waterfalls and error floors
  - Thresholds and minimum distance

<table>
<thead>
<tr>
<th></th>
<th>Turbo Codes</th>
<th>LDPC Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall performance (thresholds)</td>
<td>EXIT charts</td>
<td>Density evolution</td>
</tr>
<tr>
<td>Error floor performance (distance properties)</td>
<td>Uniform interleaver analysis</td>
<td>Pseudo- and near-codeword analysis</td>
</tr>
</tbody>
</table>
Ancient Coding Theory
Channel capacity and coding theory playing field

![Graph showing the relationship between bandwidth efficiency (η in bits/2D) and power efficiency (E_b/N_0 in dB). The graph includes a line marked 'Capacity Bound.'}]
Ancient Coding Theory
Channel capacity and coding theory playing field
Ancient Coding Theory
Channel capacity and coding theory playing field

![Diagram showing the relationship between Power Efficiency and Bandwidth Efficiency, with Capacity Bound and BPSK/QPSK capacity curves.](image)
Ancient Coding Theory

Channel capacity and coding theory playing field

![Graph showing bandwidth efficiency vs. power efficiency](image)

- **Capacity Bound**
- **BPSK/QPSK capacity**

Bandwidth Efficiency, $\eta$ (bits/2D) vs. Power Efficiency, $E_b/N_0$ (dB)
For a target bit error rate (BER) of $10^{-5}$

![Graph showing Bandwidth Efficiency, $\eta$ (bits/2D) vs Power Efficiency, $E_b/N_0$ (dB).]

- **Capacity Bound**
- **Cutoff Rate**

- BPSK/QPSK capacity
For a target bit error rate (BER) of $10^{-5}$

![Graph showing bandwidth efficiency and power efficiency](image)

- **Capacity Bound**
- **Cutoff Rate**
- **Uncoded BPSK/QPSK**
- **BPSK/QPSK capacity**

Power Efficiency, $E_b/N_0$ (dB)

Bandwidth Efficiency, $\eta$ (bits/2D)
For a target bit error rate (BER) of $10^{-5}$

- BPSK/QPSK capacity
- Uncoded BPSK/QPSK
- Hamming (7,4)

Bandwidth Efficiency, $\eta$ (bits/2D)

Power Efficiency, $E_b/N_0$ (dB)
For a target bit error rate (BER) of $10^{-5}$

Capacity Bound

BPSK/QPSK capacity

Uncoded BPSK/QPSK

Hamming (7,4)

Golay (24,12)
For a target bit error rate (BER) of $10^{-5}$

- BPSK/QPSK capacity
- Uncoded BPSK/QPSK
- BCH (255,123)
- Hamming (7,4)
- Golay (24,12)
For a target bit error rate (BER) of $10^{-5}$

Bandwidth Efficiency, $\eta$ (bits/2D)

Power Efficiency, $E_b/N_0$ (dB)

Capacity Bound

Cutoff Rate

BPSK/QPSK capacity

Uncoded BPSK/QPSK

Hamming (7,4)

BCH (255,123)

RS (64,32)

Golay (24,12)
For a target bit error rate (BER) of $10^{-5}$

![Graph showing the relationship between Power Efficiency and Bandwidth Efficiency for various coding schemes.](image)

- **Cutoff Rate**
- **Capacity Bound**
- **Uncoded BPSK/QPSK**
- **BPSK/QPSK capacity**
- **LDPC (504,3,6)**
- **BCH (255,123)**
- **RS (64,32)**
- **Golay (24,12)**
- **Hamming (7,4)**

Bandwidth Efficiency, $\eta$ (bits/2D) vs. Power Efficiency, $E_b/N_0$ (dB)
For a target bit error rate (BER) of $10^{-5}$

Bandwidth Efficiency, $\eta$ (bits/2D)

Power Efficiency, $E_b/N_0$ (dB)
For a target bit error rate (BER) of $10^{-5}$

Bandwidth Efficiency, $\eta$ (bits/2D)

Power Efficiency, $E_b/N_0$ (dB)

- Pioneer QLI (2,1,31)
- LDPC (504,3,6)
- BCH (255,123)
- RS (64,32)
- Golay (24,12)
- Hamming (7,4)
- Uncoded BPSK/QPSK
- BPSK/QPSK capacity
For a target bit error rate (BER) of $10^{-5}$

![Graph showing bandwidth efficiency vs. power efficiency for various coding schemes.](image)

- **Power Efficiency, $E_b/N_0$ (dB)**
  - Capacity Bound
  - Uncoded BPSK/QPSK
  - BPSK/QPSK capacity

- **Bandwidth Efficiency, $\eta$ (bits/2D)**
  - Pioneer QLI (2,1,31)
  - LDPC (8096,3,6)
  - LDPC (504,3,6)
  - BCH (255,123)
  - Hamming (7,4)
  - RS (64,32)
  - Golay (24,12)

For a target bit error rate of $10^{-5}$, the graph illustrates the performance of various coding schemes. The graph shows the trade-off between bandwidth efficiency and power efficiency, with different codes represented by markers on the graph. The capacity bounds and uncoded limits are also indicated, providing a benchmark for comparison.
For a target bit error rate (BER) of $10^{-5}$

<table>
<thead>
<tr>
<th>Power Efficiency, $E_b/N_0$ (dB)</th>
<th>Bandwidth Efficiency, $\eta$ (bits/2D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncoded BPSK/QPSK</td>
<td>Cutoff Rate</td>
</tr>
<tr>
<td>BPSK/QPSK capacity</td>
<td>Capacity Bound</td>
</tr>
<tr>
<td>LDPC (8096,3,6)</td>
<td></td>
</tr>
<tr>
<td>Pioneer QLI (2,1,31)</td>
<td>Hamming (7,4)</td>
</tr>
<tr>
<td>LDPC (504,3,6)</td>
<td>Voyager</td>
</tr>
<tr>
<td>Turbo (65536,18)</td>
<td>Voyager</td>
</tr>
<tr>
<td>Voyager</td>
<td>Voyager</td>
</tr>
</tbody>
</table>

- LDPC (504,3,6)
- BCH (255,123)
- RS (64,32)
- Golay (24,12)
- Hamming (7,4)
For a target bit error rate (BER) of $10^{-5}$

- Power Efficiency, $E_b/N_0$ (dB)
- Bandwidth Efficiency, $\eta$ (bits/2D)

- Capacity Bound
- Uncoded BPSK/QPSK
- BPSK/QPSK capacity

- LDPC (10^7, 3, 6)
- LDPC (8096, 3, 6)
- Pioneer QLI (2, 1, 31)
- Voyager
- Hamming (7, 4)
- Voyager
- Turbo (65536, 18)
- RS (64, 32)
- BCH (255, 123)
- Golay (24, 12)

- Cutoff Rate
(200000,3,6)-regular LDPC code

BER vs $E_b/N_0$ (dB)
(200000,3,6)-regular LDPC code

Waterfall
(200000,3,6)-regular LDPC code
(65536,18) Turbo code
Irregular LDPC code with block length $10^7$
(200000,3,6)-regular LDPC code
(65536,18) Turbo code
Irregular LDPC code with block length $10^7$

BER vs. $E_b/N_0$ (dB)

Waterfall

Error floor
For a target bit error rate of $10^{-10}$
For a target bit error rate of $10^{-10}$

Bandwidth Efficiency, $\eta$ (bits/2D)

Power Efficiency, $E_b/N_0$ (dB)

Capacity Bound

Uncoded BPSK/QPSK
For a target bit error rate of $10^{-10}$

![Graph showing the relationship between Power Efficiency ($E_b/N_0$) and Bandwidth Efficiency ($\eta$). The graph compares Capacity Bound, Cutoff Rate, Uncoded BPSK/QPSK, and Hamming (7,4) codes.]
For a target bit error rate of $10^{-10}$

Bandwidth Efficiency, $\eta$ (bits/2D)

Power Efficiency, $E_b/N_0$ (dB)

Capacity Bound

Uncoded BPSK/QPSK

BPSK/QPSK capacity

Hamming (7,4)

Golay (24,12)

Cutoff Rate

Uncoded BPSK/QPSK
For a target bit error rate of $10^{-10}$

![Diagram showing Bandwidth Efficiency vs. Power Efficiency, $E_b/N_0$ (dB), for various modulation schemes and error correction codes.](image)

- **Capacity Bound**
- **Uncoded BPSK/QPSK**
- **Hamming (7,4)**
- **BCH (255,123)**
- **Golay (24,12)**
For a target bit error rate of $10^{-10}$

![Graph showing power efficiency versus bandwidth efficiency for various codes and cutoff rates. The graph includes points for codes such as BPSK/QPSK capacity, Uncoded BPSK/QPSK, Hamming (7,4), BCH (255,123), RS (64,32), and Golay (24,12).]
For a target bit error rate of $10^{-10}$

The diagram shows the relationship between power efficiency, $E_b/N_0$ (dB), and bandwidth efficiency, $\eta$ (bits/2D), along with the capacity bounds and cutoff rate for various codes and modulation schemes.

- **Capacity Bound**
- **Uncoded BPSK/QPSK**
- **BPSK/QPSK Capacity**
- **Power Efficiency, $E_b/N_0$**
- **Bandwidth Efficiency, $\eta$ (bits/2D)**

**Codes and Modulation Schemes**
- Pioneer QLI (2,1,3)
- Hamming (7,4)
- BCH (255,123)
- RS (64,32)
- Golay (24,12)
For a target bit error rate of $10^{-10}$

![Graph showing Bandwidth Efficiency vs. Power Efficiency](image)

- Capacity Bound
- Uncoded BPSK/QPSK
- BPSK/QPSK capacity
- Pioneer QLI (2,1,3 r)
- Voyager
- Hamming (7,4)
- BCH (255,123)
- RS (64,32)
- Golay (24,12)

Power Efficiency, $E_b/N_0$ (dB)

Bandwidth Efficiency, $\eta$ (bits/2D)
For a target bit error rate of $10^{-10}$

Bandwidth Efficiency, $\eta$ (bits/2D)

Power Efficiency, $E_b/N_0$ (dB)

- BPSK/QPSK capacity
- Uncoded BPSK/QPSK
- Pioneer QLI (2,1,3)
- Turbo (65536,18)
- BCH (255,123)
- RS (64,32)
- Golay (24,12)
- Hamming (7,4)
- Voyager
For a target bit error rate of $10^{-10}$
For a target bit error rate of $10^{-10}$

![Graph showing power efficiency vs. bandwidth efficiency for various coding schemes.](image)

- **Power Efficiency, $E_b/N_0$ (dB)**
- **Bandwidth Efficiency, $\eta$ (bits/2D)**

- **Capacity Bound**
- **Uncoded BPSK/QPSK**
- **BPSK/QPSK capacity**

- **Coding Schemes**:
  - **Pioneer QLI (2,1,3)**
  - **LDPC (8096,3,6)**
  - **Turbo (65536,18)**
  - **BCH (255,123)**
  - **RS (64,32)**
  - **Voyager**
  - **LDPC (10^7) IR ???**
  - **Hamming (7,4)**
  - **Golay (24,12)**

- **Cutoff Rate**
Modern Coding Theory

TURBO Codes
Encoder 1

Encoder 2

\[ u = (u_0, u_1, \ldots, u_{K-1}) \]

\[ v^{(0)} = (v_0^{(0)}, v_1^{(0)}, \ldots, v_{K-1}^{(0)}) \]

\[ v^{(1)} = (v_0^{(1)}, v_1^{(1)}, \ldots, v_{K-1}^{(1)}) \]

\[ v^{(2)} = (v_0^{(2)}, v_1^{(2)}, \ldots, v_{K-1}^{(2)}) \]
Modern Coding Theory
Turbo Codes: Parallel concatenated codes (PCCs)

- An information block is encoded by two constituent convolutional encoders.

\[
\begin{align*}
    \mathbf{u} = (u_0, u_1, \ldots, u_{K-1}) \\
    \mathbf{v}^{(0)} = (v_0^{(0)}, v_1^{(0)}, \ldots, v_{K-1}^{(0)}) \\
    \mathbf{v}^{(1)} = (v_0^{(1)}, v_1^{(1)}, \ldots, v_{K-1}^{(1)}) \\
    \mathbf{v}^{(2)} = (v_0^{(2)}, v_1^{(2)}, \ldots, v_{K-1}^{(2)})
\end{align*}
\]
An information block is encoded by two constituent convolutional encoders.

A pseudo-random interleaver that approximates a random shuffle (permutation) is used to change the order of the data symbols before they are encoded by the second encoder.
The encoders have a recursive systematic structure.
Modern Coding Theory
Turbo Codes: Parallel concatenated codes (PCCs)

- The encoders have a recursive systematic structure.
Modern Coding Theory

Turbo Codes: Parallel concatenated codes (PCCs)

- The encoders have a recursive systematic structure.
Modern Coding Theory
Turbo Codes: Parallel concatenated codes (PCCs)

- The encoders have a recursive systematic structure.

\[ v = (v^{(0)}, v^{(1)}, v^{(2)}) \]

- The transmitted codeword is \( v = (v^{(0)}, v^{(1)}, v^{(2)}) \).
Modern Coding Theory

Turbo Codes: Iterative decoding

\[ L_A(u) \xrightarrow{\pi} L_E(u') \]

\[ \pi^{-1} \]

\[ L_E(u) \xrightarrow{\pi} L_A(u') \]

Decoder 1

Decoder 2

\[ L(\hat{u}) \]
Decoding can be done iteratively using two constituent soft-input soft-output (SISO) decoders.

For short constraint length constituent convolutional encoders, optimum SISO decoding of the constituent codes is possible using the BCJR algorithm.
Modern Coding Theory

Turbo Codes: SISO decoder

- **A priori LLRs:**
  \[ L_A(u_i) = \log \frac{\sum_{u:u_i=0} p(u)}{\sum_{u:u_i=1} p(u)} \]

- **A posteriori (APP) log-likelihood ratios (LLRs):**
  \[ L(u_i) = \log \frac{\sum_{u:u_i=0} \Lambda(u)}{\sum_{u:u_i=1} \Lambda(u)} \]
  \[ \Lambda(u) \sim p(r|u) \times p(u) \]

- **Extrinsic LLRs:**
  \[ L_E(u_i) = L(u_i) - L_A(u_i) \]
Overview of BCJR Algorithm

- The BCJR algorithm operates on the trellis representation of the convolutional code. The logarithmic version of the BCJR algorithm is preferred in practice, since it is simpler to implement and provides greater numerical stability than the probabilistic version.

- The logarithm of the path metric $\Lambda(u)$ can be written as a sum of branch metrics leaving a trellis state $s_i$ at time $i$.

$$\log \Lambda(u) = \sum_{i=0}^{K-1} \gamma(s_i, u_i)$$

$$\gamma(s_i, u_i) = \frac{u_i L_A(u_i)}{2} - \frac{E_s}{N_0} ||r_i - v_i||^2$$

- $E_s/N_0$ Channel signal-to-noise ratio
- $r_i$ Received signal vector
- $v_i$ Signal vector corresponding to symbol $u_i$
Overview of BCJR Algorithm

The likelihoods of the different states at time $i$ are determined by a forward and a backward recursion

$$\alpha(s_{i+1}) = \max^{*}(s_{i},u_{i},s_{i+1}) \in B(\rightarrow s_{i+1}) \left[ \alpha(s_{i}) + \gamma(s_{i},u_{i}) \right]$$

$$\beta(s_{i}) = \max^{*}(s_{i},u_{i},s_{i+1}) \in B(s_{i} \rightarrow) \left[ \gamma(s_{i},u_{i}) + \beta(s_{i+1}) \right]$$

with the initial conditions

$$\alpha(s_{0}) = \begin{cases} 0, & s_{0} = 0 \\ -\infty, & s_{0} \neq 0 \end{cases}$$

$$\beta(s_{K}) = \begin{cases} 0, & s_{K} = 0 \\ -\infty, & s_{K} \neq 0 \end{cases}$$

and the max operation:

$$\max^{*}[x,y] = \max[x,y] + \log(1 + \exp(-|x-y|))$$
Overview of BCJR Algorithm

- The a posteriori log likelihood ratios are then given by

\[
L(u_i) = \max^*_{(s_i, u_i, s_{i+1}) \in B(u_i=0)} \left[ \alpha(s_i) + \gamma(s_i, u_i) + \beta(s_{i+1}) \right] \\
- \max^*_{(s_i, u_i, s_{i+1}) \in B(u_i=1)} \left[ \alpha(s_i) + \gamma(s_i, u_i) + \beta(s_{i+1}) \right]
\]

- The max* operations can be replaced by the simpler max operation, resulting in the suboptimum max-log-MAP algorithm.
Summary of the (log-domain) BCJR Algorithm

- Step 1: Compute the branch metrics $\gamma(s_i, u_i)$.
- Step 2: Initialize the forward and backward metrics $\alpha(s_0)$ and $\beta(s_K)$.
- Step 3: Compute the forward metrics $\alpha(s_{i+1}), i=0,1,\ldots,K-1$.
- Step 4: Compute the backward metrics $\beta(s_i), i=K-1,K-2,\ldots,0$.
- Step 5: Compute the APP L-values $L(u_i), i=0,1,\ldots,K-1$.
- Step 6 (optional): Compute the hard decisions $\hat{u}_i, i=0,1,\ldots,K-1$. 
Example: Decoding of a 2-state recursive systematic (accumulator) convolutional code

- We use the mapping $0 \rightarrow +1$
- $1 \rightarrow -1$
- $E_s/N_0 = 1/4$
- The received vector is normalized by $\sqrt{E_s}$.
- There is no a priori information available, i.e., $L_A(u_i) = 0$.

$$r = (\begin{array}{c}
-0.8, -0.1 \\
-1, +0.5 \\
+1.8, -1.1 \\
-1.6, +1.6
\end{array})$$
Example: Decoding of a 2-state convolutional code

Computation of the branch metric

\[ \gamma(s_0 = 0, u_0 = 1) = -\frac{E_s}{N_0} ||r_i - v_i||^2 \]

\[ = -0.25 \times [(-0.8 + 1)^2 + (-0.1 + 1)^2] \]

\[ = -0.2125 \]

State transitions and branch metric computation illustrated in a trellis diagram.
Example: Decoding of a 2-state convolutional code

Computation of the forward metrics

\[ \alpha(s_2 = 0) = \max_{(s_1, u_1, s_2) \in B(s_2 = 0)} [\alpha(s_1) + \gamma(s_1, u_1)] \]

\[ = \max[(-0.2125 - 0.0625), (-1.1125 - 1.0625)] \]

\[ = -0.275 \]

\[ r = (-0.8, -0.1, -1, +0.5, +1.8, -1.1, -1.6, +1.6) \]
Example: Decoding of a 2-state convolutional code

Computation of the backward metrics

\[ \beta(s_2 = 1) = \max_{(s_2,u_2,s_3) \in B(s_2=1 \rightarrow)} [\gamma(s_2,u_2) + \beta(s_3)] \]

\[ = \max[-0.1625 - 0.18, (-3.0625 - 1.78)] \]

\[ = -0.3425 \]

\[ r = (-0.8,-0.1, -1, +0.5, +1.8, -1.1, -1.6, +1.6) \]
Example: Decoding of a 2-state convolutional code

▶ Decoding of the second bit

\[ L(u_1) = \max[(−0.2125 − 1.5625 − 0.3425), (−1.1125 − 1.0625 − 2.1425)] \]
\[ − \max[(−0.2125 − 0.0625 − 2.1425), (−1.1125 − 0.5625 − 0.3425)] \]
\[ = −2.1175 + 2.0175 = −0.1 \]

▶ Hard decision decoded sequence:

\[ \hat{u} = (0, 1, 0, 1) \]
Modern Coding Theory

Waterfall Performance: EXIT Charts
Extrinsic information transfer (EXIT) charts are used to analyze the iterative decoding behavior of the SISO constituent decoders.

EXIT charts plot the extrinsic mutual information

\[ I_E(u_i; L_E(u_i)) \]

at a SISO decoder output versus the \textit{a priori} mutual information

\[ I_A(u_i; L_A(u_i)) \]

at the decoder input, assuming the LLRs are Gaussian distributed.
Successful decoding is possible if the two curves do not cross; otherwise, decoding fails.

- The curves depend on the bit signal-to-noise ratio (SNR) $\frac{E_b}{N_0}$.
- The iterative decoding threshold of a turbo decoder is the smallest SNR for which the two curves do not cross.
An information block is first encoded by the outer encoder and the codeword \( v' \) is then interleaved to form the input \( u' \) to the inner encoder.

Again, decoding can be done iteratively using SISO decoders.
The iterative decoding threshold of an SCC can also be determined using EXIT charts.

Successful decoding is possible if the curves do not cross.

Only the EXIT curve of the inner decoder depends on the channel SNR!
Error-Floor Performance: Uniform Interleaver Analysis
The distance spectrum of a concatenated code ensemble can be analyzed using a uniform distribution of interleavers.

A uniform interleaver is a probabilistic device that maps an input block of weight d and length N into all its possible output permutations with equal probability.

The constituent encoders are decoupled by the uniform interleaver and can thus be considered to be independent.

In this way, we obtain the average Weight Enumerating Function (WEF) of the code ensemble, which represents the expected number of codewords with a given weight.
Example The average WEF of an SCC ensemble is given by:

\[
\bar{A}_{w,d}^{SCC} = \sum_{d_1=1}^{N_1} A_{w,d_1}^{outer} A_{d_1,d}^{inner} \left( \frac{N_1}{d_1} \right)
\]

The quantity \( \bar{A}_{w,d}^{SCC} \) represents the expected number of codewords of weight \( d \) that are generated by weight \( w \) input sequences.
The average WEF of the code ensemble can be used in the union bound to upper bound the bit error probability with maximum likelihood (ML) decoding as follows:

\[
P_b \leq \frac{1}{2} \sum_{d=1}^{N} \sum_{w=1}^{K} \frac{w}{K} A_{w,d} \text{erfc}\left(\sqrt{\frac{dRE_b}{N_0}}\right),
\]

where \( R \) is the code rate.

- \( P_b \) is dominated by the low weight codeword terms. If their multiplicity decreases as the interleaver size \( K \) increases, then the error probability also decreases and we say that the code exhibits **interleaver gain**.

- Iterative decoding of turbo codes behaves like ML decoding at high SNR.
Modern Coding Theory

Turbo Codes: The union bound

![Graph showing the relationship between $P_b(E)$ and $E_b/N_0$ (dB).]

- PCCC Simulation 10000
- PCCC Bound 100
- PCCC Bound 1000
- PCCC Bound 10000
- (3,1,4) Simulation

Legend:
- Solid line: PCCC Simulation 10000
- Dashed line: PCCC Bound 100
- Dotted line: PCCC Bound 1000
- Dashed-dotted line: PCCC Bound 10000
- Crosses: (3,1,4) Simulation
Asymptotic Distance Properties of Turbo Codes
Asymptotic distance properties of PCCs

- [Breiling, 2004] The minimum distance of a PCC with $K=2$ constituent encoders and block length $N$ is upper bounded by
  \[ d_{min}(N) \leq O(\ln N) \]

- [Kahale, Urbanke; 1998] The minimum distance of a multiple PCC with $K > 2$ branches grows as
  \[ d_{min}(N) = O \left( N^{\frac{K-2}{K}} \right) \]
Asymptotic distance properties of SCCs

[Kahale, Urbanke; 1998] and [Perotti, Benedetto, 2006] The minimum distance of an SCC grows as

\[ d_{\text{min}}(N) = O(N^\nu), \]

where

\[ \frac{d_{\text{min}}^C - 2}{d_{\text{min}}^C} < \nu < \frac{d_{\text{min}}^C - 1}{d_{\text{min}}^C} \]

and \( d_{\text{min}}^C \) is the minimum distance of the outer code.
Asymptotic distance properties of Multiple Serially Concatenated Codes (MSCCs)

- Parallel and single serially concatenated codes are not asymptotically good, i.e., their minimum distance does not grow linearly with block length as the block length goes to infinity.
- Multiple serially concatenated codes (MSCCs) with 3 or more serially concatenated encoders can be asymptotically good.
- The most interesting MSCCs are repeat multiple accumulate (RMA) codes, due to their simple constituent encoders.
For the ensemble of RMA codes:

- It was shown in [Pfister; 2003] that the minimum distance of RMA codes grows linearly with block length for
  \[ d_{\min}^{C_0} \geq 2 \]

- It was shown in [Pfister and Siegel; 2003] that the distance growth rate for an infinite number of accumulators meets the Gilbert-Varshamov bound (GVB).

- Distance growth rates for any finite number of accumulators were presented in [Fagnani and Ravazzi; 2008] and [Kliewer, Zigangirov, Koller, and Costello; 2008].
We observe that the minimum distance growth rates are close to the GVB. The growth rates get closer to the GVB when the number of serially concatenated component encoders or the minimum distance of the outer code increases.
Modern Coding Theory

LDPC Codes
Definition An LDPC block code is a linear block code whose parity-check matrix $H$ has a small number of ones in each row and column.

An $(N, J, K)$-regular LDPC block code is a linear block code of length $N$ and rate $R \geq 1 - J/K$ whose parity-check matrix $H$ has exactly $J$ ones in each column and $K$ ones in each row, where $J, K << N$. 
Modern Coding Theory

LDPC Codes: Parity-check matrix representation

Example The parity-check matrix $H$ of a rate $R = 1/2$, $(10,3,6)$-regular LDPC block code:

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

$H$ consists of 10 columns and 5 rows corresponding to 10 code symbols and 5 parity-check equations. Each code symbol is included in $J=3$ parity-check equations and each parity-check equation contains $K=6$ code symbols.
**Definition** A Tanner graph is a bipartite graph such that
- Each code symbol is represented by a “variable node”;
- Each parity-check equation is represented by a “check node”;
- An edge connects a variable node to a check node if and only if the corresponding code symbol participates in the corresponding parity-check equation.

- Each variable node has degree $J$ and each check node has degree $K$ for an $(N, J, K)$-regular code.
**Example** A rate $R = 1/2$, $(10,3,6)$-regular LDPC block code:

- **Parity-check matrix**

  \[
  \mathbf{H} = \begin{bmatrix}
  1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
  0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
  1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
  0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
  1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1
  \end{bmatrix}
  \]

- **Corresponding Tanner graph**
Definition A cycle of length $2L$ in a Tanner graph is a path consisting of $2L$ edges such that the start node and end node are the same.

Definition The length of the shortest cycle in a Tanner graph is called the girth.
A cycle of length 4

\[ H = \begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1
\end{bmatrix} \]
Modern Coding Theory
LDPC Codes: Cycles in Tanner graphs

A cycle of length 4

A cycle of length 6
Message-passing (belief propagation) is an iterative decoding algorithm that uses the structure of the Tanner graph.

In each iteration of the algorithm:
- Each variable node sends a message (“extrinsic information”) to each check node it is connected to;
- Each check node sends a message (“extrinsic information”) to each variable node it is connected to;
- For each code symbol (variable node), we compute the \textit{a posteriori probability} that the symbol takes on the value “1”, given all the received symbols and that all the parity-check equations are satisfied.
The most commonly used message is the log likelihood ratio (LLR) of a symbol. This is calculated using the probabilities \( p_0 \) and \( p_1 \) that the symbol takes on the values 0 and 1, respectively, and is given by \( \log(p_0/p_1) \).
The most commonly used message is the log likelihood ratio (LLR) of a symbol. This is calculated using the probabilities $p_0$ and $p_1$ that the symbol takes on the values 0 and 1, respectively, and is given by $\log(p_0/p_1)$.

At each decoding iteration
The most commonly used message is the log likelihood ratio (LLR) of a symbol. This is calculated using the probabilities $p_0$ and $p_1$ that the symbol takes on the values 0 and 1, respectively, and is given by $\log(p_0/p_1)$.

At each decoding iteration

- Each variable node computes an “extrinsic” LLR for each neighboring edge based on what was received from the “other” neighboring edges.
- The most commonly used message is the log likelihood ratio (LLR) of a symbol. This is calculated using the probabilities \( p_0 \) and \( p_1 \) that the symbol takes on the values 0 and 1, respectively, and is given by \( \log(p_0/p_1) \).

- At each decoding iteration
  - Each variable node computes an “extrinsic” LLR for each neighboring edge based on what was received from the “other” neighboring edges.
  - The variable node computation includes the incoming message from the channel.
The most commonly used message is the log likelihood ratio (LLR) of a symbol. This is calculated using the probabilities $p_0$ and $p_1$ that the symbol takes on the values 0 and 1, respectively, and is given by $\log(p_0/p_1)$.

At each decoding iteration

Each variable node computes an “extrinsic” LLR for each neighboring edge based on what was received from the “other” neighboring edges.

The variable node computation includes the incoming message from the channel.
The most commonly used message is the log likelihood ratio (LLR) of a symbol. This is calculated using the probabilities $p_0$ and $p_1$ that the symbol takes on the values 0 and 1, respectively, and is given by $\log\left(\frac{p_0}{p_1}\right)$.

At each decoding iteration

- Each variable node computes an “extrinsic” LLR for each neighboring edge based on what was received from the “other” neighboring edges.

- The variable node computation includes the incoming message from the channel.

\[ m_{5,1} = c_5 + (n_{3,5} + n_{4,5}) \]
At each decoding iteration (continued...)
At each decoding iteration (continued...)

Each check node computes an “extrinsic” LLR for each neighboring edge based on what was received from the “other” neighboring edges.
At each decoding iteration (continued...)

Each check node computes an “extrinsic” LLR for each neighboring edge based on what was received from the “other” neighboring edges.
• At each decoding iteration (continued...)

• Each check node computes an “extrinsic” LLR for each neighboring edge based on what was received from the “other” neighboring edges.

\[ n_{3,8} = \log \left( \frac{1 + \tanh \left( \frac{m_{1,3}}{2} \right) \cdot \tanh \left( \frac{m_{2,3}}{2} \right) \cdot \tanh \left( \frac{m_{4,3}}{2} \right) \cdot \tanh \left( \frac{m_{5,3}}{2} \right) \cdot \tanh \left( \frac{m_{6,3}}{2} \right)}{1 - \tanh \left( \frac{m_{1,3}}{2} \right) \cdot \tanh \left( \frac{m_{2,3}}{2} \right) \cdot \tanh \left( \frac{m_{4,3}}{2} \right) \cdot \tanh \left( \frac{m_{5,3}}{2} \right) \cdot \tanh \left( \frac{m_{6,3}}{2} \right)} \right) \]
After any number of iterations, decisions can be made on each variable node using a final update.
After any number of iterations, decisions can be made on each variable node using a final update.
After any number of iterations, decisions can be made on each variable node using a final update.

Let the LLR $d_5 = c_5 + (n_{1,5} + n_{3,5} + n_{4,5})$

- If $d_5 \geq 0$, then the variable node is decoded as a 0.
- Otherwise, the variable node is decoded as a 1.
- This assumes the mapping $0 \rightarrow +1$ and $1 \rightarrow -1$. 
The following example illustrates a simple “bit-flipping” version of message-passing decoding for the binary symmetric channel (BSC). (On a BSC, “channel errors” occur independently with probability $p < 0.5$, and the transmitted bits are received correctly with probability $1 - p$.)

Received (corrupted) codeword

\[ y_1 + y_2 + y_3 \quad y_1 + y_3 + y_4 \quad y_2 + y_3 + y_4 \]
\[ 1 + y_2 + y_3 \quad 1 + y_3 + y_4 \quad y_2 + y_3 + y_4 \]
$1 + 0 + y_3 \quad 1 + y_3 + y_4 \quad 0 + y_3 + y_4$
1 + 0 + 0
1 + 0 + y_4
0 + 0 + y_4
\[
\begin{align*}
1 + 0 + 0 &= 1 + 0 + 1 \\
0 + 0 + 1 &= 1
\end{align*}
\]
\[1 + 0 + 0 = 1\]
\[1 + 0 + 1 = 0\]
\[0 + 0 + 1 = 1\]
\[
\begin{align*}
1 + 0 + 0 &= 1 \\
1 + 0 + 1 &= 0 \\
0 + 0 + 1 &= 1
\end{align*}
\]
Flip,  

1 + 0 + 0  = 1  

0 + 0 + 1  = 1  

1 + 0 + 1  = 0  

0 + 0 + 0  = 1
1 + 0 + 0 = 1
1 + 0 + 1 = 0
0 + 0 + 1 = 1
1
Flip, Stay

0
Flip, Flip

0
Flip, Stay, Flip

1
Stay, Flip

1 + 0 + 0
= 1

1 + 0 + 1
= 0

0 + 0 + 1
= 1
Flip, Stay
1 + 0 + 0 = 1

Flip, Flip
1 + 0 + 1 = 0

Flip, Stay, Flip
0 + 0 + 1 = 1

Stay, Flip
\[ y_1 + y_2 + y_3 \quad y_1 + y_3 + y_4 \quad y_2 + y_3 + y_4 \]
\[ 1 + y_2 + y_3 \quad 1 + y_3 + y_4 \quad y_2 + y_3 + y_4 \]
\[ 1 + 1 + 0 \]

\[ 1 + 0 + y_4 \]

\[ 1 + 0 + y_4 \]
\begin{align*}
1 + 1 + 0 &= 1 \\
1 + 0 + 1 &= 0 \\
1 + 0 + 1 &= 1
\end{align*}
\[1 + 1 + 0 \quad 1 + 0 + 1 \quad 1 + 0 + 1\]

\[= 0\]
\[1 + 1 + 0 = 0\]
\[1 + 0 + 1 = 0\]
\[1 + 0 + 1 = 0\]

Diagram:

- Three nodes labeled 1, 1, 0, and 1 are connected by lines.

- Calculations:
  - \[1 + 1 + 0 = 0\]
  - \[1 + 0 + 1 = 0\]
1 + 1 + 0 = 0
1 + 0 + 1 = 0
1 + 0 + 1 = 0
1 + 1 + 0 = 0
1 + 0 + 1 = 0
1 + 0 + 1 = 0

Stay, Stay, Stay,
1 + 1 + 0 = 0
1 + 0 + 1 = 0
1 + 0 + 1 = 0
Stay, Stay
Stay, Stay, Stay,
Stay, Stay

Stay, Stay

Stay, Stay, Stay

Stay, Stay

$1 + 1 + 0 = 0$

$1 + 0 + 1 = 0$

$1 + 0 + 1 = 0$
Decoded codeword

Stay, Stay
Stay, Stay
Stay, Stay, Stay
Stay, Stay

1 + 1 + 0 = 0
1 + 0 + 1 = 0
1 + 0 + 1 = 0
The author acknowledges Dr. Lara Dolecek’s permission to use this example in this talk.
Waterfall Performance: Density Evolution
Observation As the block length $N$ of an LDPC block code increases, its performance on a BPSK modulated AWGN channel improves - up to a limit.

This limit is called the iterative decoding threshold and it depends on the channel quality, specified in terms of the SNR for an AWGN channel, and the “degree profile” of the code.

If the SNR falls below the threshold, then iterative decoding will fail, and the BER will be bounded away from zero.

Example The iterative decoding threshold for $(3,6)$-regular LDPC block codes is $E_b/N_0=1.11$ dB (0.93 dB from capacity).
Comparing the performance of (3,6)-regular LDPC block codes with the (3,6) iterative decoding threshold and the rate $R=1/2$ BPSK AWGN channel capacity.
Messages exchanged over the Tanner graph are random variables with probability density functions (PDFs) that evolve at each iteration.

Density evolution is a technique used to compute the thresholds of LDPC block code ensembles.

The channel SNR is fixed and the PDFs of the messages exchanged over the Tanner graph are tracked.

The lowest channel SNR at which the PDFs “converge” to the correct values is called the iterative decoding threshold.

* The Tanner graph is assumed to be cycle-free, i.e., it corresponds to a very long (infinite length) code with very large (infinite) girth.
By varying the “degree profiles” (the distribution of variable and check node degrees), irregular LDPC code ensembles with better iterative decoding thresholds than regular ensembles can be found.

Global optimization techniques can be used in conjunction with density evolution to find optimum degree profiles that minimize iterative decoding thresholds.
LDPC Codes: Irregular LDPC block codes

(200000,3,6)-regular LDPC code
Irregular LDPC code with block length 240000

BER vs. $E_b/N_0$ (dB) graph

Threshold

$E_b/N_0$ (dB)
Modern Coding Theory
LDPC Codes: Protograph-based codes

- Regular or irregular ensembles of LDPC block codes can be constructed from a projected graph (protograph) using a copy-and-permute operation.
An ensemble of protograph-based LDPC block codes is obtained by placing random permutors on each edge of the protograph.

- The copy-and-permute operation preserves the degree profile of a Tanner graph.
- Since the iterative decoding threshold is a function of the degree profile only, an ensemble of protograph-based block codes has the same threshold as the underlying protograph.
A (time-varying) convolutional code is the set of sequences $\mathbf{v}$ satisfying the equation $\mathbf{v} H_{\text{conv}}^T = \mathbf{0}$, where the parity-check matrix $H_{\text{conv}}$ is given by

$$H_{\text{conv}} = \begin{bmatrix}
\vdots \\
H_{m_s}(t) & \cdots & H_0(t) \\
H_{m_s}(t+1) & \cdots & H_0(t+1) \\
\vdots \\
H_{m_s}(t+m_s) & \cdots & H_0(t+m_s) \\
\vdots
\end{bmatrix}$$

$H_{\text{conv}}^T$ is the syndrome former (transposed parity-check) matrix and, for a rate $R = b/c$ code, the elements $H_i(t)$, $i = 0, 1, \cdots, m_s$, are binary $(c - b) \times c$ submatrices.

The value $m_s$, called the syndrome former memory, is determined by the maximal width (in submatrices) of the nonzero area in the matrix $H_{\text{conv}}$. 
A (time-varying) convolutional code is the set of sequences $\mathbf{v}$ satisfying the equation $\mathbf{v}^T \mathbf{H}_{\text{conv}} = \mathbf{0}$, where the parity-check matrix $\mathbf{H}_{\text{conv}}$ is given by

$$
\mathbf{H}_{\text{conv}} = \begin{bmatrix}
\cdots & \cdots & \cdots & \cdots & \cdots \\
\mathbf{H}_{m_s}(t) & \mathbf{H}_{m_s}(t+1) & \mathbf{H}_0(t) & \mathbf{H}_0(t+1) & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
\mathbf{H}_{m_s}(t+m_s) & \cdots & \mathbf{H}_0(t+m_s) & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots 
\end{bmatrix}
$$

The constraint length $\nu_s$ is equal to the maximal width (in symbols) of the nonzero area in the matrix $\mathbf{H}_{\text{conv}}$ and is given by $\nu_s = (m_s + 1)c$.

If the submatrices $\mathbf{H}_i(t) = \mathbf{H}_i(t + P)$, $i = 0, 1, \cdots, m_s$, $\forall t$, the code is periodically time-varying with period $P$.

If the submatrices $\mathbf{H}_i(t)$, $i = 0, 1, \cdots, m_s$, do not depend on $t$, the code is time-invariant.
The parity-check matrix of an \((\nu_s, J, K)\) -regular low-density parity-check (LDPC) convolutional code has exactly \(K\) ones in each row and \(J\) ones in each column of \(H_{\text{conv}}\), where \(J, K << \nu_s\).

An LDPC convolutional code is called irregular if its row and column weights are not constant.
Modern Coding Theory
LDPC Codes: Convolutional counterparts

- (10,3,6) time-varying convolutional code \((m_s = 4, c = 2, b = 1, R = 1/2)\)

\[ H_{\text{conv}} = \begin{bmatrix}
\cdots & 1 & 1 & 1 & 1 & 1 & 1 \\
\cdots & 1 & 1 & 0 & 1 & 1 & 1 \\
\cdots & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots \\
1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots 
\end{bmatrix} \]
Graph Structure of LDPC Convolutional Codes

- Time-invariant LDPC convolutional code with rate $R = 1/3$ and $m_s = 2$.

- Graph has infinitely many nodes and edges.

- Node connections are localized within one constraint length.
Taking advantage of the localized structure of the graph, \( I \) identical, independent processors can be pipelined to perform \( I \) iterations in parallel.

The pipeline decoder operates on a finite window of length \( \nu_S \cdot I \) received symbols that slides along the received sequence.

Once an initial decoding delay (latency) of \( \nu_S \cdot I \) received symbols has elapsed, the decoder produces a continuous output stream.
For terminated transmission over an AWGN channel, regular LDPC convolutional codes have been shown to achieve better iterative decoding thresholds than their block code counterparts:

\[
\begin{array}{|c|c|c|}
\hline
(J, K) & (E_b/N_0)_{conv} & (E_b/N_0)_{block} \\
\hline
(3,6) & 0.46 \text{ dB} & 1.11 \text{ dB} \\
(4,8) & 0.26 \text{ dB} & 1.61 \text{ dB} \\
(5,10) & 0.21 \text{ dB} & 2.04 \text{ dB} \\
\hline
\end{array}
\]

[Lentmaier, Sridharan, Zigangirov, Costello, ISIT 2005]
Error-Floor Performance: Pseudo- and Near-Codeword Analysis
The requirements of capacity-approaching code performance can be summarized as follows:

Capacity-approaching performance
The requirements of capacity-approaching code performance can be summarized as follows:

- Capacity-approaching performance
- Very large block length
The requirements of capacity-approaching code performance can be summarized as follows:

- Capacity-approaching performance
- Very large block length
- Iterative decoding
The requirements of capacity-approaching code performance can be summarized as follows:

- Sparse code representation
- Capacity-approaching performance
- Very large block length
- Iterative decoding
The requirements of capacity-approaching code performance can be summarized as follows:

- Sparse code representation
- Iterative decoding
- Very large block length
- Capacity-approaching performance

*
The requirements of capacity-approaching code performance can be summarized as follows:

- Good sparse codes are typically designed using EXIT charts or density evolution. These codes exhibit capacity-approaching performance in the waterfall region. However, they can have much worse performance when lower BER values are required, i.e., they can exhibit error floors.
Error floors of (3,6)-regular LDPC codes for block lengths between 2048 and 8096.

[T. Richardson, “Error floors of LDPC codes,” Allerton 2003]
The error floor behavior observed in LDPC code performance curves is commonly attributed both to properties of the code and the sparse parity-check representation that is used for decoding.

These include:

- pseudo-codewords (due to locally operating decoding algorithms)
- near-codewords (due to trapping sets)
- poor asymptotic distance properties (effects can be seen in the ML decoding union bound)
Message-passing iterative decoders for LDPC codes are known to be subject to decoding failures due to pseudo-codewords. These failures can cause the high channel SNR performance to be worse than that predicted by the ML decoding union bound, resulting in the appearance of error floors.

The low complexity of message-passing decoding comes from the fact that the algorithm operates locally on the Tanner graph representation.

However, since it works locally, the algorithm cannot distinguish if it is acting on the graph itself, or some finite cover of the graph.

Pseudo-codewords are defined as codewords in codes corresponding to graph covers.
Consider the same 3-cover of the protograph given earlier, this time with the nodes reordered so that similar nodes are grouped together.
Consider the following assignment of values to variable nodes:

```
  1  0  0  1  0  0  0  0  0  1  0  0
```

```plaintext
Consider the following assignment of values to variable nodes
```
Consider the following assignment of values to variable nodes:

1 0 0
1 0 0
0 0 0
1 0 0
Consider the following assignment of values to variable nodes:

({1,0,0},{1,0,0},{0,0,0},{1,0,0}) is a valid codeword in the cubic cover. Therefore \( \omega = \left( \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3} \right) \) is a valid pseudo-codeword of the original graph.
With message-passing decoding of LDPC codes, error floors are usually observed when the decoding algorithm converges to a near-codeword, i.e., a sequence that satisfies most of the parity-check equations, but not all of them.

This behavior is usually attributed to so-called trapping sets in the Tanner graph representation.

[T. Richardson, “Error floors of LDPC codes,” Allerton 2003]
An \((a,b)\)-generalized trapping set is a set of “\(a\)” variable nodes such that the induced subgraph includes “\(b\)” check nodes of odd degree.

**Example** Consider the following \((3,1)\)-generalized trapping set
An \((a,b)\)-generalized trapping set is a set of “\(a\)” variable nodes such that the induced subgraph includes “\(b\)” check nodes of odd degree.

**Example** Consider the following \((3,1)\)-generalized trapping set
An \((a,b)\)-generalized trapping set is a set of “\(a\)” variable nodes such that the induced subgraph includes “\(b\)” check nodes of odd degree.

**Example** Consider the following \((3,1)\)-generalized trapping set:
An \((a,b)\)-generalized trapping set is a set of “\(a\)” variable nodes such that the induced subgraph includes “\(b\)” check nodes of odd degree.

**Example** Consider the following \((3,1)\)-generalized trapping set

If these three variable nodes are “erroneously” received, it is hard for the decoding algorithm to converge to their correct values, since only one of the neighboring check nodes can *detect* a problem.
Asymptotic Distance Properties of LDPC Codes
Randomly constructed code ensembles usually have asymptotically good distance properties, i.e., their minimum distance grows linearly with block length as the block length goes to infinity.

The ensemble of linear block codes meets the Gilbert-Varshamov bound, the best known bound on asymptotic distance growth rate.

Sparse codes, on the other hand, do not have asymptotically good distance properties in general.

Regular LDPC codes usually are asymptotically good, whereas only some ensembles of irregular LDPC codes possess this property.
Regularity and Expansivity of LDPC Codes

Modern Coding Theory
Asymptotic distance properties

Asymptotic distance properties of LDPC block codes

- Regular LDPC code ensembles are asymptotically good, i.e., their minimum distance grows linearly with block length [Gallager, 1962].
- Expander codes (generalized LDPCs) are also asymptotically good if the expanded check nodes (subcodes) have \( d_{\text{min}} \geq 3 \) [Barg, Zémor, 2004].
- Some irregular LDPC code ensembles can be asymptotically good [Litsyn, Shevelev; 2003] and [Di, Richardson, Urbanke, 2006].
- Some irregular LDPC code ensembles based on protographs can also be asymptotically good [Divsalar, Jones, Dolinar, Thorpe; 2005].
Asymptotic distance properties of LDPC codes

- For an ensemble of LDPC block codes:
  \[ d_{\text{min}} \geq \delta_{\text{min}} N \text{ as } N \to \infty \Rightarrow \text{asymptotically good codes.} \]

- For an ensemble of LDPC convolutional codes:
  \[ d_{\text{free}} \geq \delta_{\text{free}} \nu_s \text{ as } \nu_s \to \infty \Rightarrow \text{asymptotically good codes.} \]

- For regular ensembles, LDPC convolutional codes can achieve larger distance growth rates than their block code counterparts.

<table>
<thead>
<tr>
<th>(J, K)</th>
<th>( \delta_{\text{min}} )</th>
<th>( \delta_{\text{free}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3,6)</td>
<td>0.023</td>
<td>0.083</td>
</tr>
<tr>
<td>(4,8)</td>
<td>0.063</td>
<td>0.191</td>
</tr>
</tbody>
</table>

[Sridharan, Truhachev, Lentmaier, Costello, Zigangirov, IT Trans. 2007]
Asymptotic distance properties of protograph-based LDPC convolutional codes

- The random permutation representation of protograph-based LDPC block codes allows the uniform interleaver analysis to be used to compute their asymptotic distance growth rates [Divsalar, 2006].

- This technique can be extended to analyze ensembles of unterminated, protograph-based, periodically time-varying LDPC convolutional codes.

- The free distance growth rates of the LDPC convolutional code ensembles exceed the minimum distance growth rates of the corresponding LDPC block code ensembles [Mitchell et al., 2008].
Block code growth rate $\delta_{\text{min}}$
Convolutional code growth rate $\delta_{\text{free}}$

$d_{\text{min}}/N$ or $d_{\text{free}}/\nu_s$