Basics of Error Control Codes

Source:
*Information Theory, Inference, and Learning Algorithms*
David MacKay
© Cambridge Univ. Press 2003

Downloadable or purchasable:
Channel coding aka Forward Error Correction

- “My communication system is working, but I am getting a lot of errors…what can I do?”
- CRC is an error DETECTING code…it spots errors with high probability, but doesn’t tell you how to fix them
- Error CORRECTING codes can actually allow you to repair the errors…if there aren’t too many
Channel coding is adding redundancy to improve reliability, at a cost in rate
  - Error correction

Source coding is removal of redundancy from information bits to improve rate
  - Compression

This lecture is only about channel coding
How do error correcting codes work?

- Basic idea: add redundancy (extra bits) to make communication more robust
  - Or, put another way, don’t allow all bit patterns, just a subset…if you receive an invalid bit sequence, correct to the closest valid bit sequence

- The extra bits (or disallowed bit patterns) reduce the net communication rate:
  - If “information bits” are denoted $i$ and “error correction bits” denoted $c$, then the new rate, with error correction is $i/(i+c)$
  - The original rate, with no error correction ($c=0$) is 1.0
## Noisy communication channels

<table>
<thead>
<tr>
<th>Transmit side</th>
<th>Channel / noise model</th>
<th>Receive side</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optical modem</td>
<td>airgap</td>
<td>Optical modem</td>
</tr>
<tr>
<td>EF modem</td>
<td>airgap</td>
<td>EF modem</td>
</tr>
<tr>
<td>modem</td>
<td>phone line</td>
<td>modem</td>
</tr>
<tr>
<td>wifi AP</td>
<td>radio waves</td>
<td>wifi client</td>
</tr>
<tr>
<td>Galileo probe</td>
<td>radio waves</td>
<td>Earth</td>
</tr>
<tr>
<td>Parent cell</td>
<td>daughter cell 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>daughter cell 2</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>disk drive</td>
<td>RAM</td>
</tr>
<tr>
<td>RAM</td>
<td>flash memory</td>
<td>RAM</td>
</tr>
<tr>
<td>printer</td>
<td>QR code</td>
<td>phone camera</td>
</tr>
<tr>
<td>Server</td>
<td>Internet</td>
<td>client</td>
</tr>
</tbody>
</table>
A model for the noise in the channel

- Binary Symmetric Channel (BSC) with $f=0.1$
  - $f$: probability of bit flip

  $\begin{align*}
  P(y=0|x=0) &= 1 - f; \\
  P(y=0|x=1) &= f; \\
  P(y=1|x=0) &= f; \\
  P(y=1|x=1) &= 1 - f.
  \end{align*}$

Other important channels: **Erasure Channel** (models packet loss in wired or wireless networks)
Example 1: Repetition code, “R3”

<table>
<thead>
<tr>
<th>Received codeword</th>
<th>Decoded as</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0 (no errors)</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>1 (no errors)</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
</tr>
</tbody>
</table>

- Each 1 information bit gets encoded to 3 transmitted bits, so the rate of this code is 1/3
- If you think of the first bit as the message, and bits 2 and 3 as the error correction bits, then the rate also turns out to be $1/(1+2) = 1/3$
- This code can correct 1 bit flip, or 2 bit erasures (erasures not shown)
Problems with R3

Noise set to flip 10% of the bits
Rate is only 1/3
Still 3% errors remaining after error correction…Crummy!
## Example 2: Random code

<table>
<thead>
<tr>
<th>Original message</th>
<th>Codewords transmitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>10100110</td>
</tr>
<tr>
<td>001</td>
<td>11010001</td>
</tr>
<tr>
<td>010</td>
<td>01101011</td>
</tr>
<tr>
<td>011</td>
<td>00011101</td>
</tr>
<tr>
<td>100</td>
<td>01101000</td>
</tr>
<tr>
<td>101</td>
<td>11001010</td>
</tr>
<tr>
<td>110</td>
<td>10111010</td>
</tr>
<tr>
<td>111</td>
<td>00010111</td>
</tr>
</tbody>
</table>

Each block of 3 info bits mapped to a random 8 bit vector...rate 3/8 code. Could pick any rate, since we just pick the length of the random code words. Note that we are encoding blocks of bits (length 3) jointly

Problems with this scheme:
(1) the need to distribute and store a large codebook
(2) decoding requires comparing received bit vectors to entire codebook
An error correcting code selects a subset of the space to use as valid messages (codewords). Since the number of valid messages is smaller than the total number of possible messages, we have given up some communication rate in exchange for robustness. The size of each ball above gives approximately the amount of redundancy. The larger the ball (the more redundancy), the smaller the number of valid messages.
The name of the game

- In ECCs is to find mathematical schemes that allow time- and space-efficient encoding and decoding, while providing high communication rates and low bit error rates, despite the presence of noise.
Types of ECC

- **Algebraic**
  - Hamming Codes
  - Reed-Solomon [CD, DVD, hard disk drives, QR codes]
  - BCH

- **Sparse graph codes**
  - Turbo [CDMA2000 1x]
  - Repeat accumulate
  - LDPC (Low Density Parity Check)
    - [WiMax, 802.11n, 10GBase 10 802.3an]
  - Fountain / Tornado / LT / Raptor (for erasure)
    - [3GPP mobile cellular broadcast, DVB-H for IP multicast]
Other ECC terminology

- Block vs. convolutional ("stream")
- Linear
  - Encoding can be represented as matrix multiply
- Systematic / non-Systematic
  - Systematic means original information bits are transmitted unmodified.
    - Repetition code is systematic
    - Random code is not (though you could make a systematic version of a random code...append random check bits that don't depend on the data...would be harder to decode than computed parity bits...would the systematic random code perform better?)
Example 3: (7,4) Hamming Code (Encoding)

Don’t encode 1 bit at a time, as in the repetition code
Encode blocks of 4 source bits to blocks of 7 transmitted

\[ s_1s_2s_3s_4 \rightarrow t_1t_2t_3t_4t_5t_6t_7 \]

Where \( t_1 \) - \( t_4 \) are chosen s.t.

\[ s_1s_2s_3s_4 \rightarrow s_1s_2s_3s_4 \ mod \ 2 \]

Set parity check bits \( t_5 \) – \( t_7 \) using

\[ t_5 = s_1 + s_2 + s_3 \ mod \ 2 \rightarrow 1 + 0 + 0 = 1 \]
\[ t_6 = s_2 + s_3 + s_4 \ mod \ 2 \rightarrow 0 + 0 + 0 = 0 \]
\[ t_7 = s_1 + s_3 + s_4 \ mod \ 2 \rightarrow 1 + 0 + 0 = 1 \]

Parity check bits are a linear function information bits— a linear code

b—example:
\[ 1000 \rightarrow 1000101 \]

Rate 4/7 code

David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003
Example 3: (7,4) Hamming Code (Encoding)

The 16 codewords of the (7,4) Hamming code:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s</code></td>
<td><code>t</code></td>
<td><code>s</code></td>
<td><code>t</code></td>
<td><code>s</code></td>
<td><code>t</code></td>
<td><code>s</code></td>
</tr>
<tr>
<td>0000</td>
<td>0000000</td>
<td>0100</td>
<td>0100110</td>
<td>1000</td>
<td>1000101</td>
<td>1100</td>
</tr>
<tr>
<td>0001</td>
<td>0001011</td>
<td>0101</td>
<td>0101101</td>
<td>1001</td>
<td>1001110</td>
<td>1101</td>
</tr>
<tr>
<td>0010</td>
<td>0010111</td>
<td>0110</td>
<td>0110001</td>
<td>1010</td>
<td>1010010</td>
<td>1110</td>
</tr>
<tr>
<td>0011</td>
<td>0011100</td>
<td>0111</td>
<td>0111010</td>
<td>1011</td>
<td>1011001</td>
<td>1111</td>
</tr>
</tbody>
</table>

Any pair of codewords differs in at least 3 bits!
Example 3: (7,4) Hamming Code (Encoding)

Since it is a linear code, we can write the encoding operation as a matrix multiply (using mod 2 arithmetic):

\[ t = G^T s \]

where

\[
G^T = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
\end{bmatrix},
\]

G is called the Generator Matrix of the code.
Example 3: (7,4) Hamming Code (Decoding)

If received vector $r = t + n$ (transmitted plus noise), then write $r$ in circles:

- Transmit example

Compute parity for each circle (dash $\rightarrow$ violated parity check)
Pattern of parity checks is called the “syndrome”
Error bit is the unique one inside all the dashed circles

Dashed line $\rightarrow$ parity check violated
* $\rightarrow$ flip that one to correct

3 possible received msgs, each due to different errors

David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003
Example 3: (7,4) Hamming Code (Decoding)

Each of the 3 circles is either dotted (syndrome=1) or solid (syndrome = 0)  
\[ 2^3 = 8 \text{ possibilities} \]

<table>
<thead>
<tr>
<th>Syndrome $z$</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unflip this bit</td>
<td>none</td>
<td>$r_7$</td>
<td>$r_6$</td>
<td>$r_4$</td>
<td>$r_5$</td>
<td>$r_1$</td>
<td>$r_2$</td>
<td>$r_3$</td>
</tr>
</tbody>
</table>
What happens if there are 2 errors?

*s denote actual errors
Circled value is incorrectly inferred single-bit error
Optimal single-bit decoder actually adds another error in this case…so we started with 2 errors and end with 3
Larger (7,4) Hamming example

7% errors remain after error correction
Comparing codes

Binary symmetric channel with $f = 0.1$

Error probability $p_b$ vs communication rate $R$ for repetition codes, (7,4) Hamming code, BCH codes up to length 1023

David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003
What is the best a code can do?

- How much noise can be tolerated?
- What SNR do we need to communicate reliably?
- At what rate can we communicate with a channel with a given SNR?
  - What error rate should we expect?
What is the best a code can do?

- Binary symmetric channel with $f = 0.1$

$$R = \frac{C}{1 - H_2(p_b)} \quad \text{where}$$

$$C = 1 - H_2(f)$$

$$H_2(p) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

$p_b$: error probability we can achieve

(best possible codes lie in this direction)

David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003
Better codes
Example 3: (7,4) Hamming Code

Don't encode 1 bit at a time, as in the repetition code
Encode blocks of 4 source bits to blocks of 7 transmitted
\[ s_1s_2s_3s_4 \rightarrow t_1t_2t_3t_4t_5t_6t_7 \]
Where \( t_1 \) - \( t_4 \) are chosen s.t.
\[ s_1s_2s_3s_4 \rightarrow s_1s_2s_3s_4 t_5t_6t_7 \]
Set parity check bits \( t_5 \) – \( t_7 \) using
\[ t_5 = s_1 + s_2 + s_3 \mod 2 \rightarrow 1+0+0 = 1 \]
\[ t_6 = s_2 + s_3 + s_4 \mod 2 \rightarrow 0+0+0 = 0 \]
\[ t_7 = s_1 + s_3 + s_4 \mod 2 \rightarrow 1+0+0 = 1 \]
Parity check bits are a linear function information bits...a linear code

Rate 4/7 code

---

Don’t encode 1 bit at a time, as in the repetition code
Encode blocks of 4 source bits to blocks of 7 transmitted
\[ s_1s_2s_3s_4 \rightarrow t_1t_2t_3t_4t_5t_6t_7 \]
Where \( t_1 \) - \( t_4 \) are chosen s.t.
\[ s_1s_2s_3s_4 \rightarrow s_1s_2s_3s_4 t_5t_6t_7 \]
Set parity check bits \( t_5 \) – \( t_7 \) using
\[ t_5 = s_1 + s_2 + s_3 \mod 2 \rightarrow 1+0+0 = 1 \]
\[ t_6 = s_2 + s_3 + s_4 \mod 2 \rightarrow 0+0+0 = 0 \]
\[ t_7 = s_1 + s_3 + s_4 \mod 2 \rightarrow 1+0+0 = 1 \]
Parity check bits are a linear function information bits...a linear code

---

Don’t encode 1 bit at a time, as in the repetition code
Encode blocks of 4 source bits to blocks of 7 transmitted
\[ s_1s_2s_3s_4 \rightarrow t_1t_2t_3t_4t_5t_6t_7 \]
Where \( t_1 \) - \( t_4 \) are chosen s.t.
\[ s_1s_2s_3s_4 \rightarrow s_1s_2s_3s_4 t_5t_6t_7 \]
Set parity check bits \( t_5 \) – \( t_7 \) using
\[ t_5 = s_1 + s_2 + s_3 \mod 2 \rightarrow 1+0+0 = 1 \]
\[ t_6 = s_2 + s_3 + s_4 \mod 2 \rightarrow 0+0+0 = 0 \]
\[ t_7 = s_1 + s_3 + s_4 \mod 2 \rightarrow 1+0+0 = 1 \]
Parity check bits are a linear function information bits...a linear code

---

b--example:
1000→1000101

Rate 4/7 code
Example 3: (7,4) Hamming Code (Encoding)

Since it is a linear code, we can write the encoding operation as a matrix multiply (using mod 2 arithmetic):

\[ t = G^T s \]

where

\[ G^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \]

\( G \) is called the Generator Matrix of the code.
Matrix formulation

Define $P$ s.t.

$$G^T = \begin{bmatrix} I_4 \\ P \end{bmatrix}$$

If

$$H = \begin{bmatrix} P & I_3 \end{bmatrix}$$

then syndrome $z = Hr$

All codewords $t = G^Ts$ satisfy

$$H^t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$H^t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

**Proof:**

$$HG^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \pmod{2}$$

$G$: Generator matrix…for encoding

$P$: “Parity check” portion of $G$ and $H$

$H$: Parity check matrix…for decoding
Graphical representation of (7,4) Hamming code

- Bipartite graph --- two groups of nodes...all edges go from group 1 (circles) to group 2 (squares)
- Circles: bits
- Squares: parity check computations
End
Low Density Parity Check Codes

- Invented in Gallagher’s MIT PhD Thesis 1960
- Computationally intractable at the time
- Re-invented by David MacKay & Radford Neal in the 1990s

**H**

Same (small) number of 1s in each row (4) and column (3)

Each row of H corresponds to a check (square)
Each col of H is a bit (circle)

As in the Hamming code example before, encode using \( t = G^T s \)
Decode involves checking parity by multiplying \( H r \), where \( r \) is a column vector of received bits

*David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003*
Decoding

- Ideal decoders would give good performance, but optimally decoding parity check codes is an NP-complete problem.
- In practice, the sum-product algorithm, aka iterative probabilistic decoding, aka belief propagation do very well.
- Decoding occurs by message passing on the graph...same basic idea as graphical models.
  - Same algorithms were discovered simultaneously in the 90s in AI / Machine Learning / Coding.
  - Decoding is an inference problem: infer likeliest source message given received message, which is the corrupted-encoded-source-message.
Pause to recall two decoding perspectives

- **Encode:** \( t = G^T s \)
- **Transmission:** \( r = t + n \)
- **Decoding:** find \( s \) given \( r \)
  - **Codeword decoding**
    - Iterate to find \( x \) close to \( r \) s.t. \( H \cdot x = 0 \) … then hopefully \( x = t \), and \( n = r - x \)
  - **Syndrome decoding**
    - Compute syndrome \( z = H \cdot r \)
    - Iterate to find \( n \) s.t. \( H \cdot n = z \)
      - We actually want \( H \cdot (t + n) = z \), but \( H(t + n) = H \cdot t + H \cdot n = 0 + H \cdot n = z \)
      - In other words, \( Hn = z \rightarrow H(t + n) = z \) [because knowing \( n \) and \( r \rightarrow t \)]
Why are we covering this?

- It’s important recent research that is transforming the landscape of communicating with embedded (and other) devices…the benefit of studying at a research university is being exposed to the latest research.
- Iterative decoding / belief propagation / sum-product algorithm techniques are also useful in many other contexts (e.g. machine learning, inference), so it’s good to be exposed to them.
- CS needs more of this kind of content.
- Q: But aren’t these algorithms impossible to implement on the embedded micros we’re focusing on?
  - A1: Encoding is easy…all you need is enough memory. Asymmetric architectures (tiny wireless embedded device talking to giant cloud server) are becoming increasingly important. LDPC codes are a good fit for architectures like this. Figuring out how to do LDPC encoding on REALLY tiny processors, with tiny amounts of memory, is an interesting research question!
  - A2: In a few years you won’t be able to buy a micro so small it won’t have the horsepower to do LDPC _decoding_.
  - A3: You could actually implement LDPC decoding using a network of small embedded devices.
Binary Erasure Channel (BEC) example

See Siegel deck
Iterative decoding of LDPC codes

Noise level: 7.5%

K=10000 info bits
N=20000 transmit bits

Shannon limit for a rate ½ code: ~11%

BER < 1/30000
BER < 3.3 x10^-5

Error-free after 13 iterations

http://www.inference.phy.cam.ac.uk/mackay/codes/gifs/

David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003
How to decode

- Propagate probabilities for each bit to be set around the graph (cf belief propagation in AI)
  - The difficulty is the cycles in the graph...So...pretend there are no cycles and iterate
- In horizontal step (from point of view of parity check matrix H) find \( r \), prob of observed parity check value arising from hypothesized bit settings
- In vertical step, find \( q \), prob of bit settings, assuming hypothesized
Iterative decoding --- Matlab example
End
How to decode

$m$ indexes checks
$n$ indexes bits

$r_{012}^0$ is the message from check 1 to variable 2. The message tells bit 2 what check 1 thinks bit 2’s value should be (i.e. the prob that bit 2 should be 0)

$r_{mn}$: parity check probabilities

Received data: $x = [0\ 1\ 0\ 0\ 0\ 0\ 0]$  
z computed from rcv data: $[1\ 1\ 0]$  
All bit hypotheses for neighbors of $m=1$ check & node 2 excluded: “-”

Hypothesized [list of hypotheses summed over in $r_{012}^0$ calc]

\[
\begin{align*}
0000 & \Rightarrow P(z1=1|x2=0,w=000) = 0 \\
0001 & \Rightarrow P(z1=1|x2=0,w=001) = 1 \\
0010 & \Rightarrow P(z1=1|x2=0,w=010) = 1 \\
0011 & \Rightarrow P(z1=1|x2=0,w=011) = 0 \\
1000 & \Rightarrow P(z1=1|x2=0,w=100) = 1 \\
1001 & \Rightarrow P(z1=1|x2=0,w=101) = 0 \\
1010 & \Rightarrow P(z1=1|x2=0,w=110) = 0 \\
1011 & \Rightarrow P(z1=1|x2=0,w=111) = 1
\end{align*}
\]
How to decode

m indexes checks
n indexes bits

$q_{12}$ is the message *from* variable 2 *to* check 1. The message tells check 1 what variable 2 thinks check 1’s value should be.

$q_{mn}$: variable probabilities

Received data: $x = [0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]$

$z$ computed from rcv data: $[1 \ 1 \ 0]$
How to decode

\( r_{mn} \): parity check probabilities

\( q_{mn} \): bit probabilities (for each edge)

\[
I_{mn} = \sum_{\{x_n: n' \in N(m) \setminus n\}} P(z_m | x_n = 0, \{x_{n'}: n' \in N(m) \setminus n\})
\]

Value of syndrome (directly determined by data!)

Hypothesized bit settings

How consistent is a hypothetical bit vector with the syndrome? Value of this term is either 0 or 1.

\[
0(i,jp) = 0(i,jp) + PzGivenX0*qp
\]

where

\( qp = 1.0; \) % "q product"...a product should start at 1!
for \( b = 0:rweight-1; \) % For each bit, i.e. each variable node we're connected to
  \( jp = ind(b+1); \) % \( jp \) gets actual index of current bit \( b \) (+1: Matlab starts at 1)
  \( qp = qp*( (1-hyp(b+1))*q0(i,jp) + hyp(b+1)*q1(i,jp)); \)
    % \( hyp(b+1) \) indicates whether bit we're looking at is a 0 or 1...
    % depending on the value of \( hyp \), we'll need to get our prob from either \( q0 \) or \( q1 \)
end

and where

\( PzGivenX0 = 1-mod(bitsum0+z(i),2); \) % This is either 0 or 1
\( PzGivenX1 = 1-mod(bitsum1+z(i),2); \) % This should also = 1-\( PzGivenX0 \)
How to decode

- $m$ indexes checks
- $n$ indexes bits
- $N(m)$ means all the bits connected to check $m$
- $n'$ in $N(m) \setminus n$ means every bit associated with check $m$, EXCEPT for bit $n$

$r_{mn}^0$: parity check probabilities
$q_{mn}$: bit probabilities (for each edge)

Value of observed syndrome (this is data!)
Hypothesized bit settings
Approx. probability of the hypothesized bit settings

$$r_{mn}^0 = \sum_{\{x_n: n' \in N(m) \setminus n\}} P(\overline{z_m | x_n = 0}, \{x_{n'}: n' \in N(m) \setminus n\}) \prod_{n' \in N(m) \setminus n} \frac{x_{n'}}{q_{mn}}$$

1 or 0, depending on whether observed syndrome $z_m$ is consistent with hypothesis to the right of $|$
How to decode

m indexes checks
n indexes bits

M(n) means all the checks connected to bit n
m' in M(n) \ m means every check associated with bit n, EXCEPT for check m

$q_{mn}$: bit probabilities, for each edge

\[
q_{mn}^0 = \alpha_{mn} p_n^0 \prod_{m' \in M(n) \setminus m} r_{m'n}^0
\]

\[
q_{mn}^1 = \alpha_{mn} p_n^1 \prod_{m' \in M(n) \setminus m} r_{m'n}^1
\]
Iterative decoding of LDPC codes

Noise level: 7.5%

K=10000 info bits
N=20000 transmit bits

Shannon limit for a rate $\frac{1}{2}$ code: $\sim$11%

BER < $\frac{1}{30000}$
BER < $3.3 \times 10^{-5}$

Error-free after 13 iterations

http://www.inference.phy.cam.ac.uk/mackay/codes/gifs/
Performance of LDPC codes
Fountain codes

- AKA Raptor codes, LT codes, rateless codes, etc
- LDPC codes for the erasure channel
  - Packet loss…useful for broadcast channels
- Instead of regular LDPC (same number of 1s in each row, or same number of edges between checks and variables), *irregular* LDPC: a few check nodes with many edges, most check nodes with just a few edges
- Irregular LDPC seems to only work well for erasure channel…error floor problem
Convolutional codes

- Can use LFSR as encoder!

Rate 1/3 non-recursive, non-systematic convolutional encoder, constraint length 3
Convolutional codes

- Can use LFSR as encoder!

Rate 1/2 recursive, systematic convolutional encoder, constraint length 4
Decoding convolutional codes

- Trellis diagram for first code
- Solid lines: 0 is input
- Dashed lines: 1 is input
- Decoding: use Viterbi algorithm for max likelihood estimate, feasible for small codes