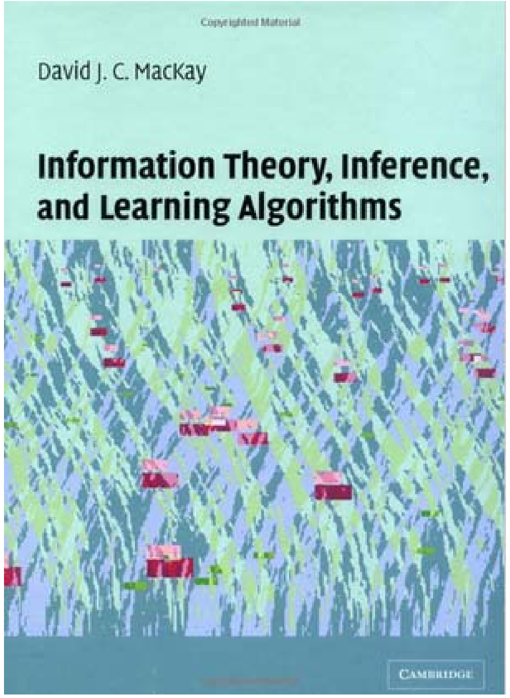
Basics of Error Control Codes

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

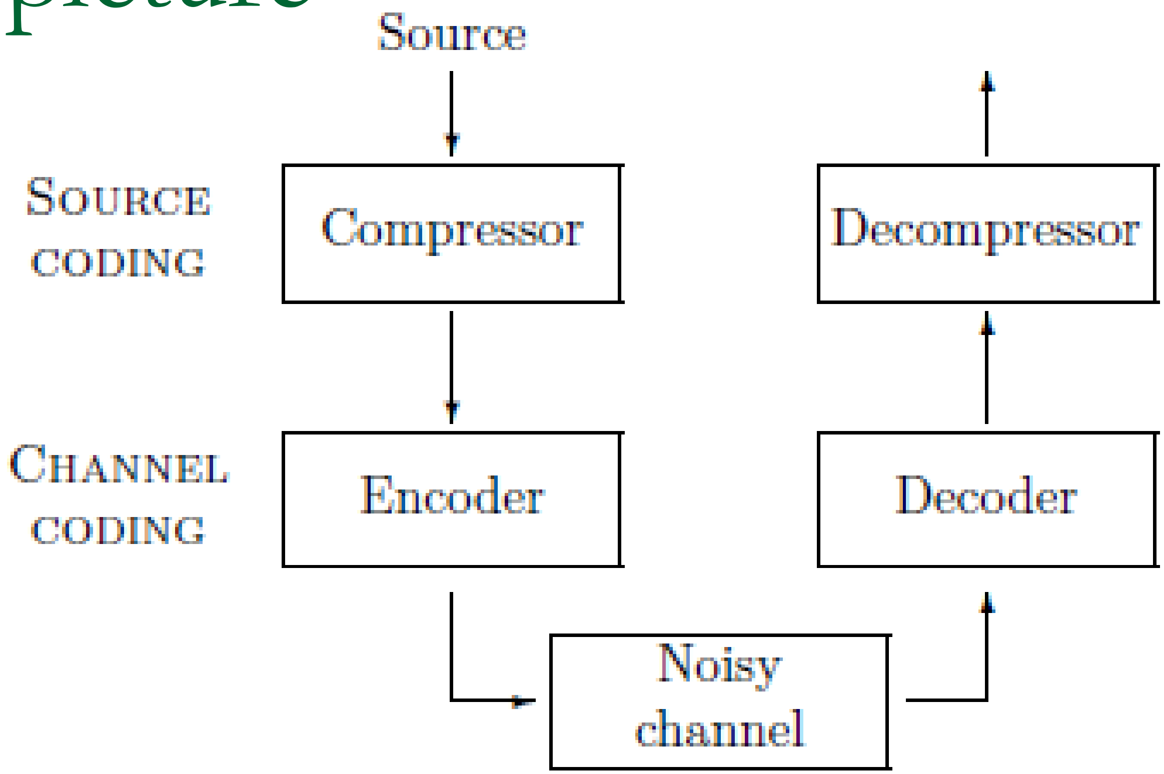
Downloadable or purchasable:
http://www.inference.phy.cam.ac.uk/mackay/itila/book.html



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

The big picture



- 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

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

How do error correcting codes work?

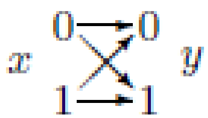
- Basic idea: add redundancy (extra bits) to make communication more robust
  - Or, put another way, don't allow all bit patterns, just a subset…if you receive an invalid bit sequence, correct to the closest valid bit sequence
- The extra bits (or disallowed bit patterns) reduce the net communication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

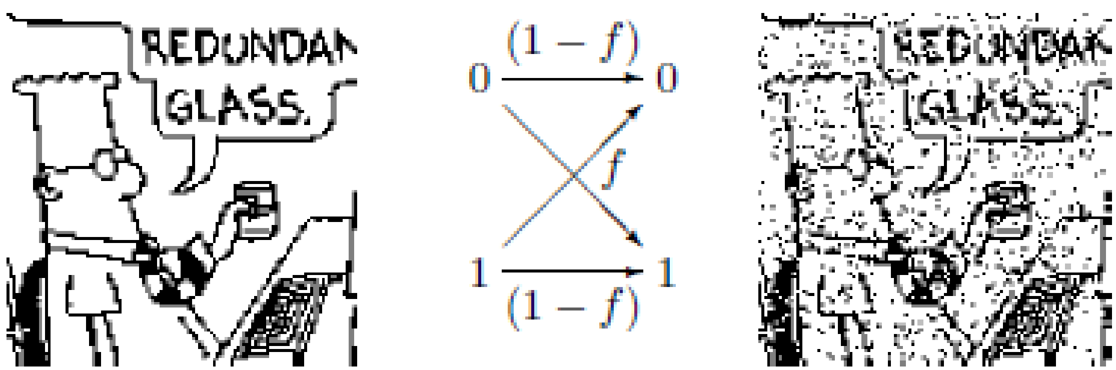
Transmit side		Channel / noise model		Receive side
Optical modem	→	airgap	→	Optical modem
EF modem	→	airgap	→	EF modem
modem	→	phone line	→	modem
wifi AP	→	radio waves	→	wifi client
Galileo probe	→	radio waves	→	Earth
Parent cell	→	daughter cell 1		
	→	daughter cell 2		
RAM	→	disk drive	→	RAM
RAM	→	flash memory	→	RAM
printer	→	QR code	→	phone camera
Server	→	Internet	→	client

A model for the noise in the channel

- Binary Symmetric Channel (BSC) with *f*=0.1
  - *f*: probability of bit ***f***lip



$$
\begin{aligned}
P(y=0 \mid x=0) &= 1-f; & P(y=0 \mid x=1) &= f; \\
P(y=1 \mid x=0) &= f; & P(y=1 \mid x=1) &= 1-f.
\end{aligned}
$$



Other important channels: ***Erasure Channel*** (models packet loss in wired or wireless networks)

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

Example 1: Repetition code, “R3”

Received codeword	Decoded as
000	0 (no errors)
001	0
010	0
100	0
111	1 (no errors)
110	1
101	1
011	1

- 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

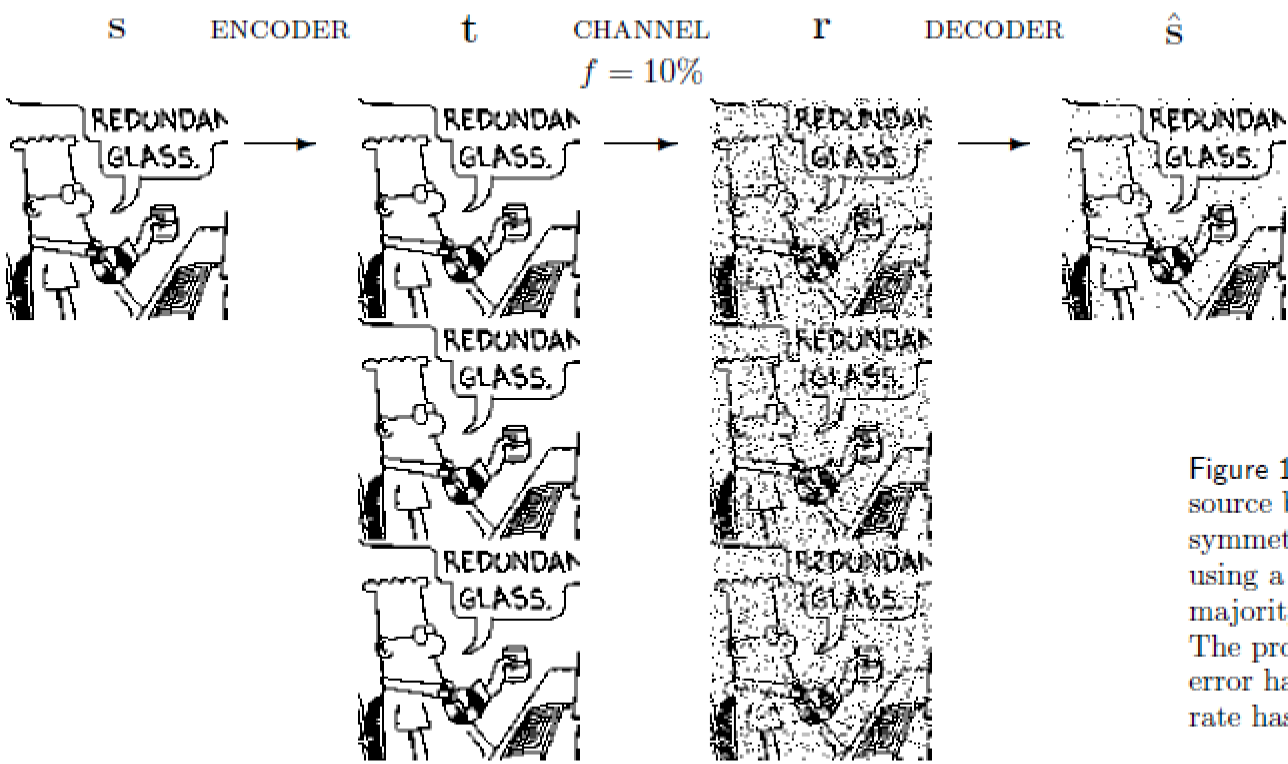


Figure 1.11. Transmitting 10 000 source bits over a binary symmetric channel with $f = 10\%$ using a repetition code and the majority vote decoding algorithm. The probability of decoded bit error has fallen to about 3%; the rate has fallen to 1/3.

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

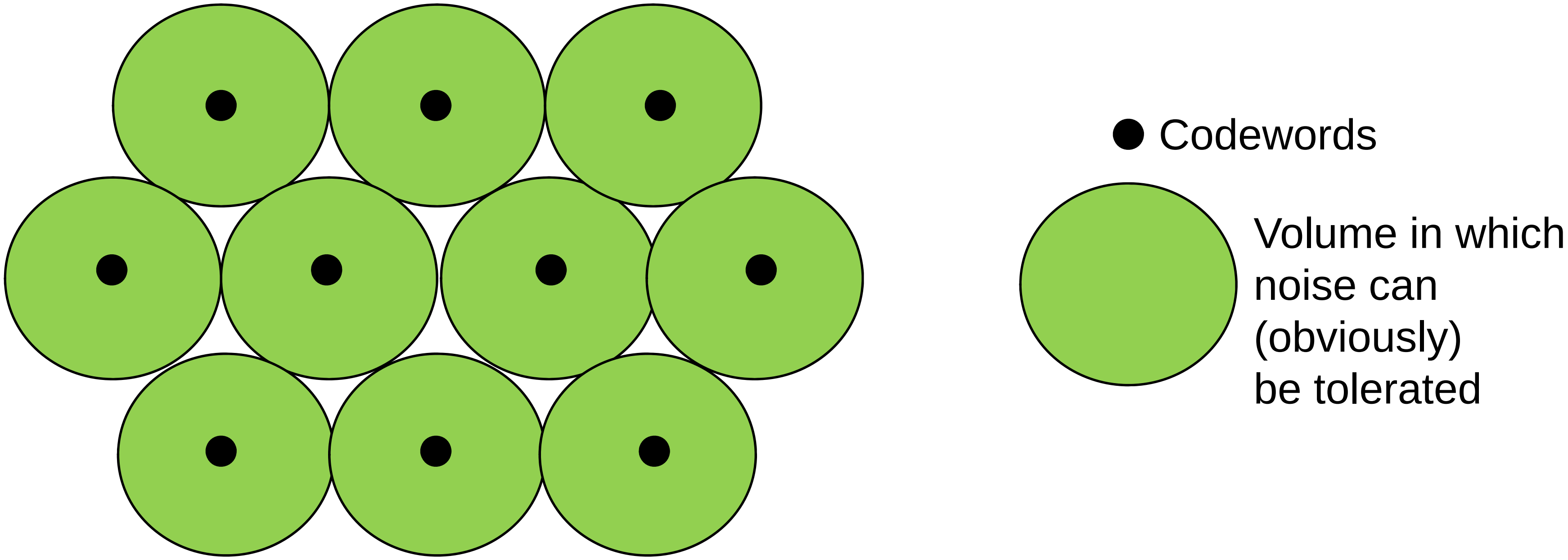
David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003

Example 2: Random code

Original message	Codewords transmitted
000	10100110
001	11010001
010	01101011
011	00011101
100	01101000
101	11001010
110	10111010
111	00010111

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

A visualization of ECCs



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

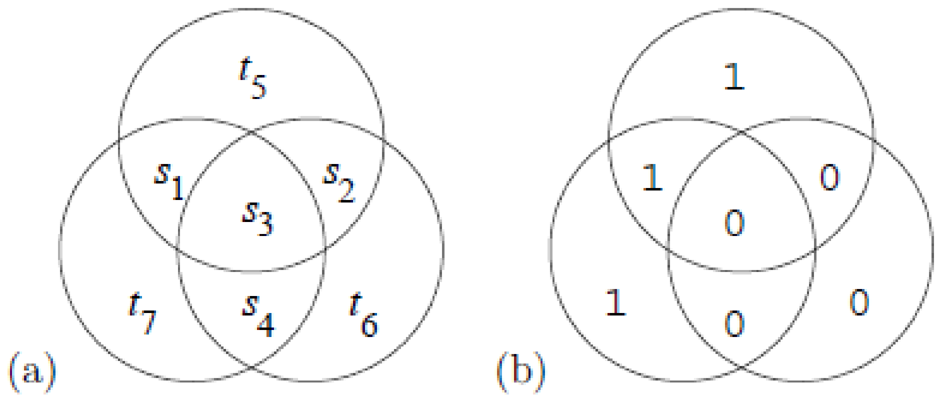
Types of ECC

- 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)



b--example:
1000→1000101

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

$$s_1 s_2 s_3 s_4 \rightarrow t_1 t_2 t_3 t_4 t_5 t_6 t_7$$

Where t_1 - t_4 are chosen s.t.

$$s_1 s_2 s_3 s_4 \rightarrow s_1 s_2 s_3 s_4\, t_5 t_6 t_7$$

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*

David MacKay
Information Theory, Inference, and Learning Algorithms

Example 3: (7,4) Hamming Code (Encoding)

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

s	t	s	t	s	t	s	t
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

Any pair of codewords differs in at least 3 bits!

David MacKay
Information Theory, Inference, and Learning Algorithms

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):

$\mathbf{t}=\mathbf{G}^{\mathsf{T}}\mathbf{s}$ where

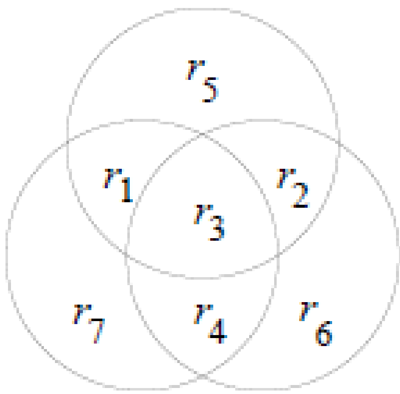
$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix},$$

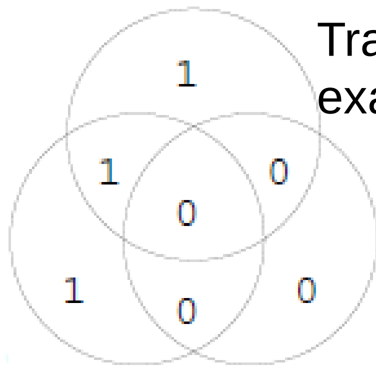
G is called the Generator Matrix of the code.

David MacKay
Information Theory, Inference, and Learning Algorithms

Example 3: (7,4) Hamming Code (Decoding)

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





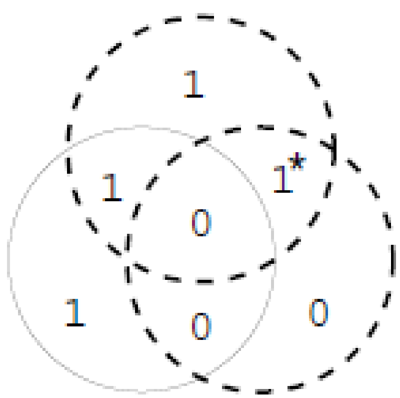
Transmitted example

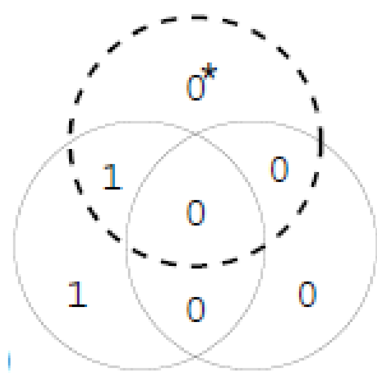
David MacKay
Information Theory, Inference, and Learning Algorithms
© Cambridge Univ. Press 2003

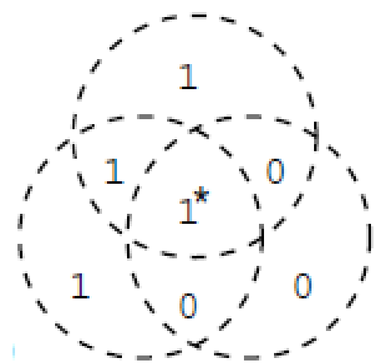
Compute parity for each circle (dash➔violated parity check)
Pattern of parity checks is called the "syndrome"
Error bit is the unique one inside all the dashed circles

Dashed line➔parity check violated
* ➔flip that one to correct

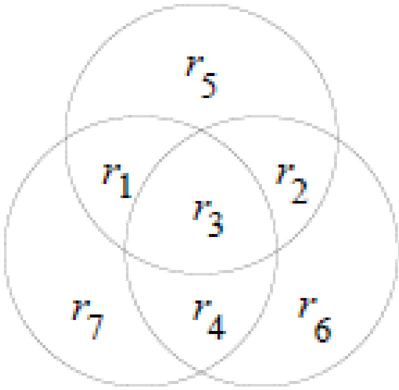
3 possible received msgs, each due to different errors







Example 3: (7,4) Hamming Code (Decoding)

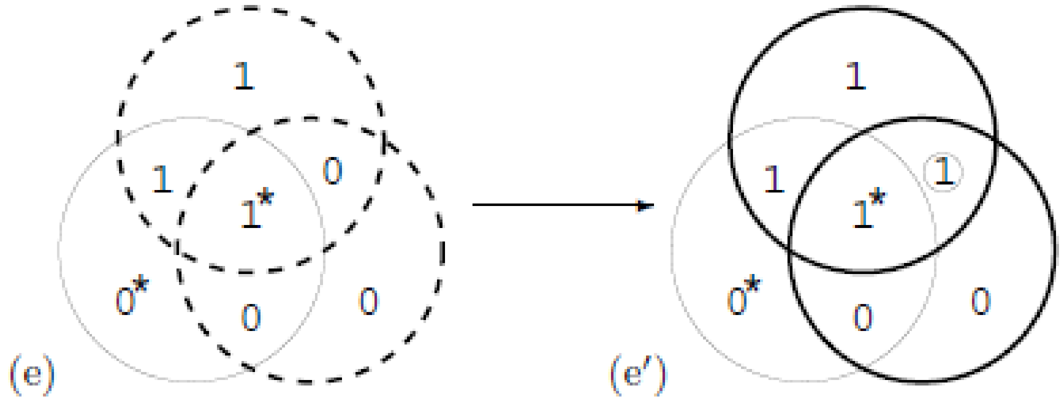


Each of the 3 circles is either dotted (syndrome=1) or solid (syndrome = 0)
➔ 2^3=8 possibilities

Syndrome z	000	001	010	011	100	101	110	111
Unflip this bit	*none*	r_7	r_6	r_4	r_5	r_1	r_2	r_3

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

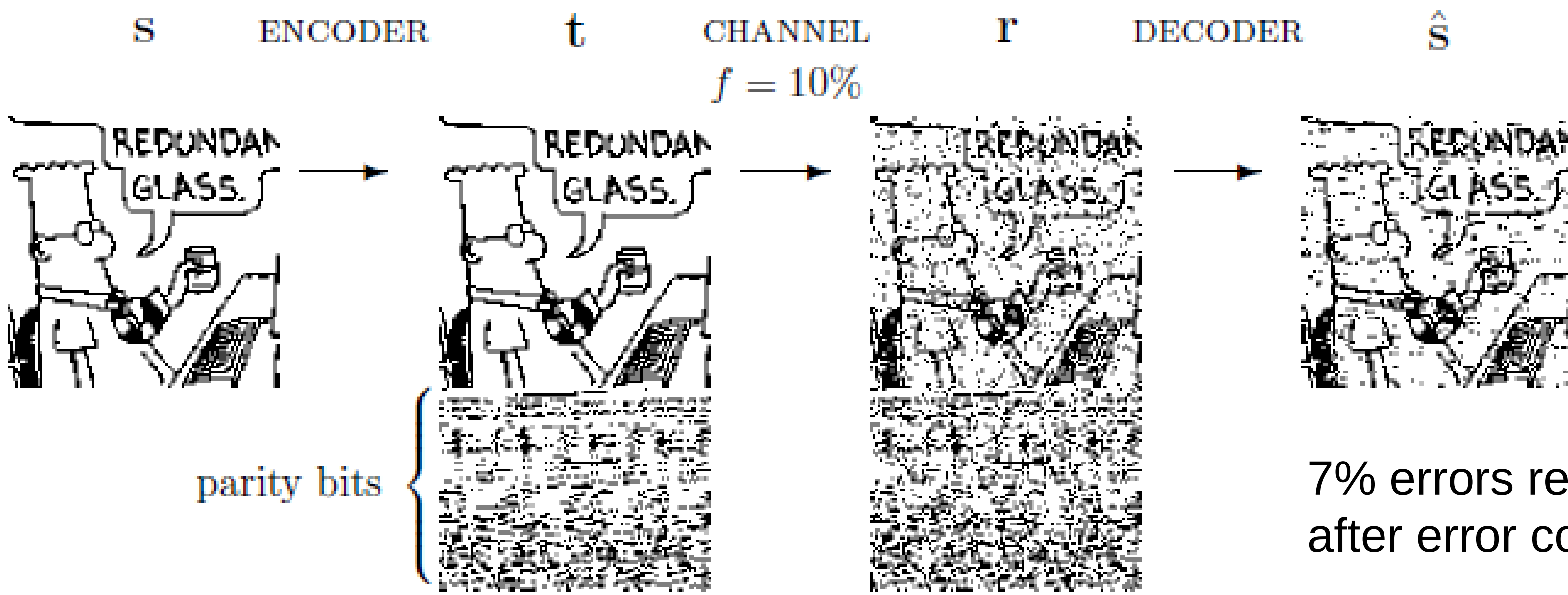
What happens if there are 2 errors?



*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

David MacKay
Information Theory, Inference, and Learning Algorithms

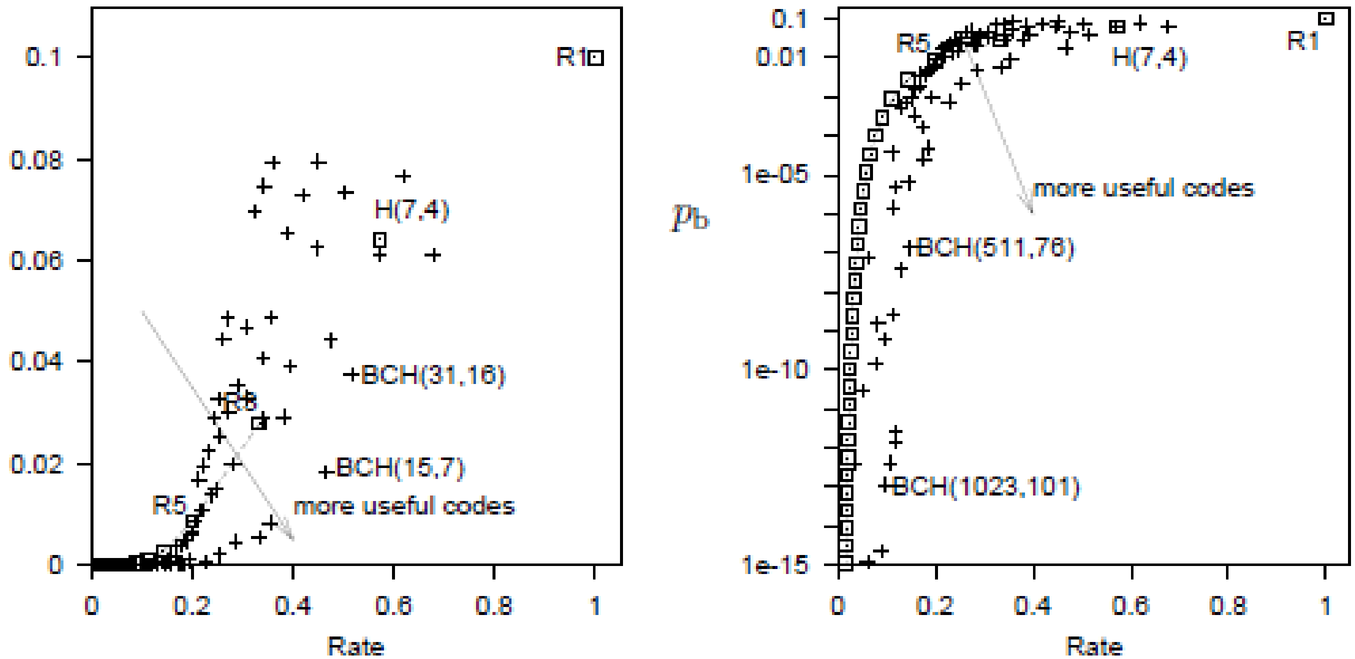
Larger (7,4) Hamming example



7% errors remain
after error correction

David MacKay
Information Theory, Inference, and Learning Algorithms

Comparing codes



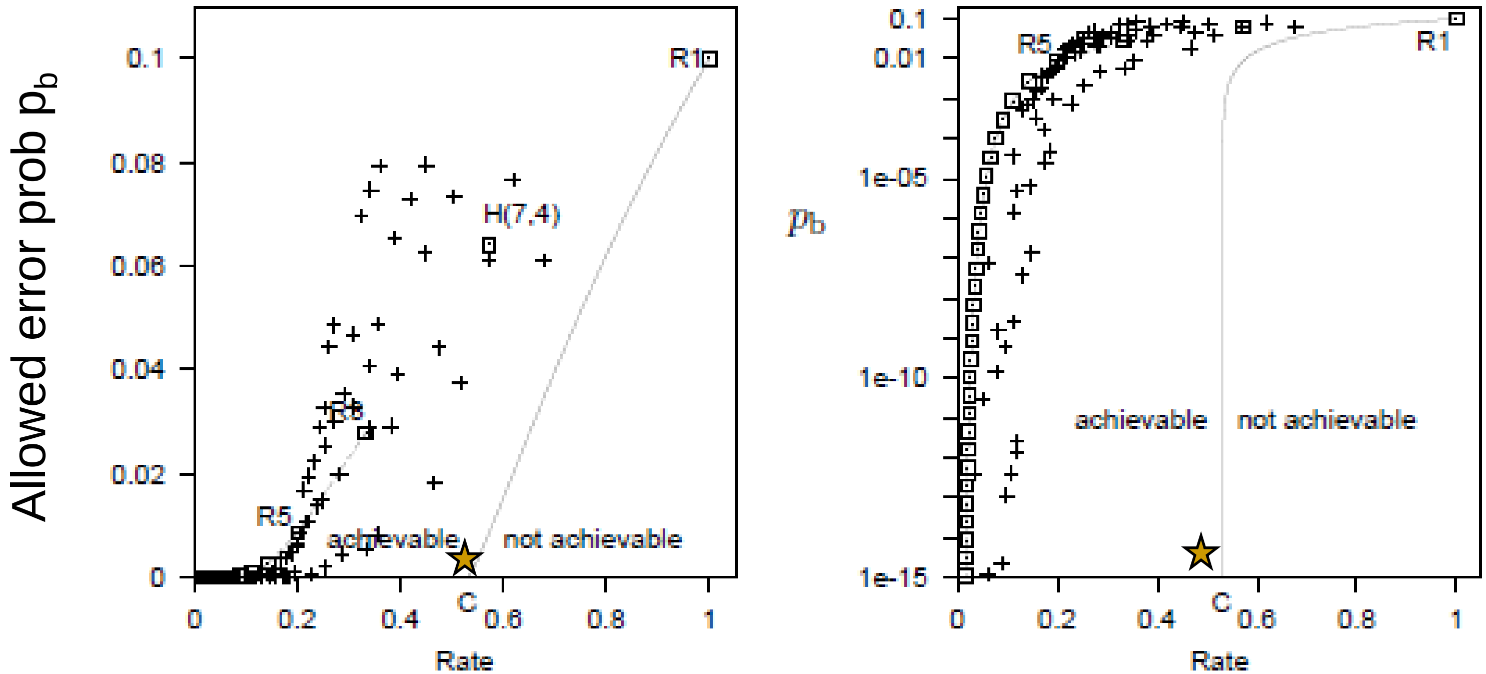
Binary symmetric channel with $f = 0.1$
Error probability p_b vs communication rate R for repetition codes, (7,4) Hamming code, BCH codes up to length 1023

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

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

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

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

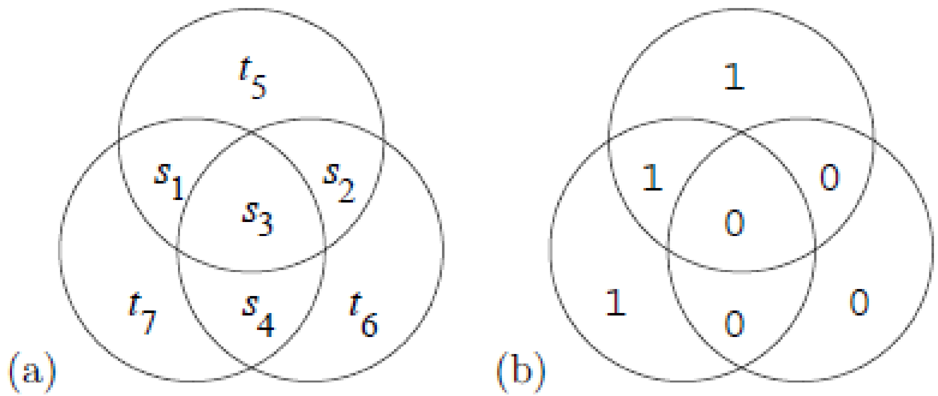
p_b: error probability we can achieve

★ Best possible codes lie in this direction

David MacKay
Information Theory, Inference, and Learning Algorithms

Better codes

Example 3: (7,4) Hamming Code (Encoding)



b--example:
1000→1000101

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*

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

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):

$\mathbf{t}=\mathbf{G}^{\mathsf{T}}\mathbf{s}$ where

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix},$$

G is called the Generator Matrix of the code.

David MacKay
Information Theory, Inference, and Learning Algorithms

Matrix formulation

Define $\mathbf{P}$ s.t.

$$\mathbf{G}^T = \begin{bmatrix} \mathbf{I}_4 \\ \mathbf{P} \end{bmatrix} \quad \text{If}$$

$$\mathbf{G}^T = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \\ 1 & 1 & 1 & \\ & 1 & 1 & 1 \\ 1 & & 1 & 1 \end{bmatrix}$$

$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_3 \end{bmatrix}$ then syndrome $\mathbf{z} = \mathbf{Hr}$

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

All codewords $\mathbf{t} = \mathbf{G}^T \mathbf{s}$ satisfy

$$\mathbf{Ht} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

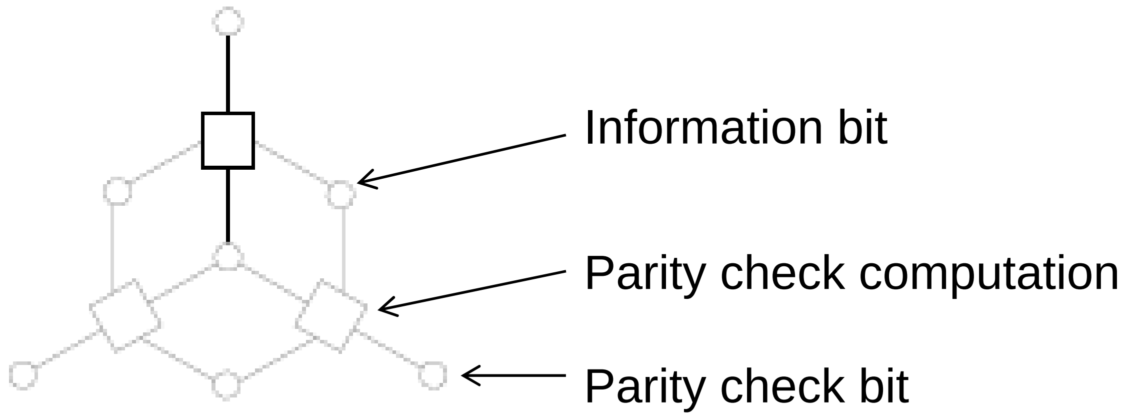
Proof: $\mathbf{HG}^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \bmod 2$

G: Generator matrix…for encoding

H: Parity check matrix…for decoding

P: “Parity check” portion of G and H

Graphical representation of (7,4) Hamming code

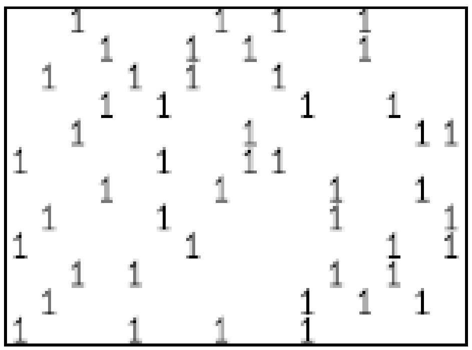


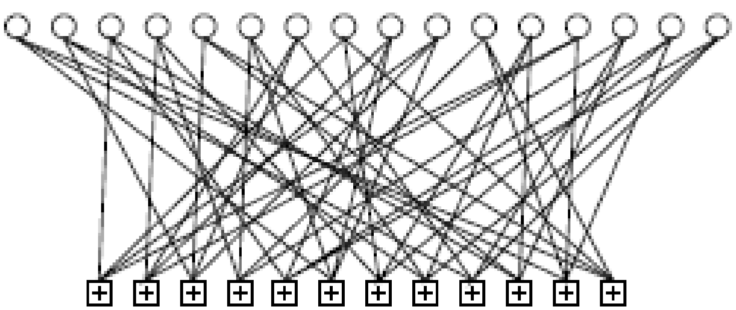
- Bipartite graph --- two groups of nodes…all edges go from group 1 (circles) to group 2 (squares)
- Circles: bits
- Squares: parity check computations

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

$\mathbf{H} =$ 



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 $\mathbf{t}=\mathbf{G}^T\mathbf{s}$
Decode involves checking parity by multiplying $\mathbf{H}\,\mathbf{r}$, where $\mathbf{r}$ 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: $\mathbf{t} = \mathbf{G}^T\mathbf{s}$
- Transmission: $\mathbf{r} = \mathbf{t}+\mathbf{n}$
- Decoding: find **s** given **r**
- <u>Codeword</u> decoding
  - Iterate to find **x** close to **r** s.t. **H x** = 0 … then hopefully **x** = **t**, and **n** = **r** - **x**
- <u>Syndrome</u> decoding
  - Compute syndrome **z** = **H r**
  - Iterate to find **n** s.t. **Hn** = **z**
    - We actually want **H** (**t+n**) = **z**, but **H**(**t**+**n**) = **Ht** + **Hn** = 0 + **Hn = z**
    - In other words, **Hn=z** ➔ **H(t+n)=z** [because knowing **n** and **r** ➔ **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

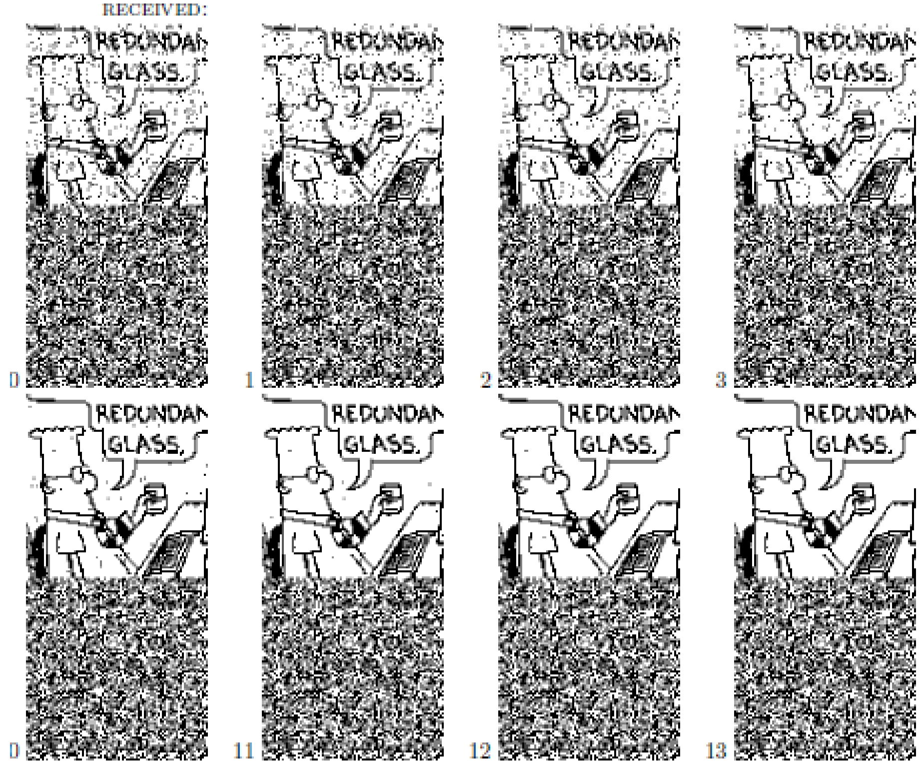
Iterative decoding of LDPC codes

Noise level: 7.5%

K=10000 info bits
N=20000 transmit bits

Shannon limit for a rate ½ code: ~11%

RECEIVED:



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

DECODED:

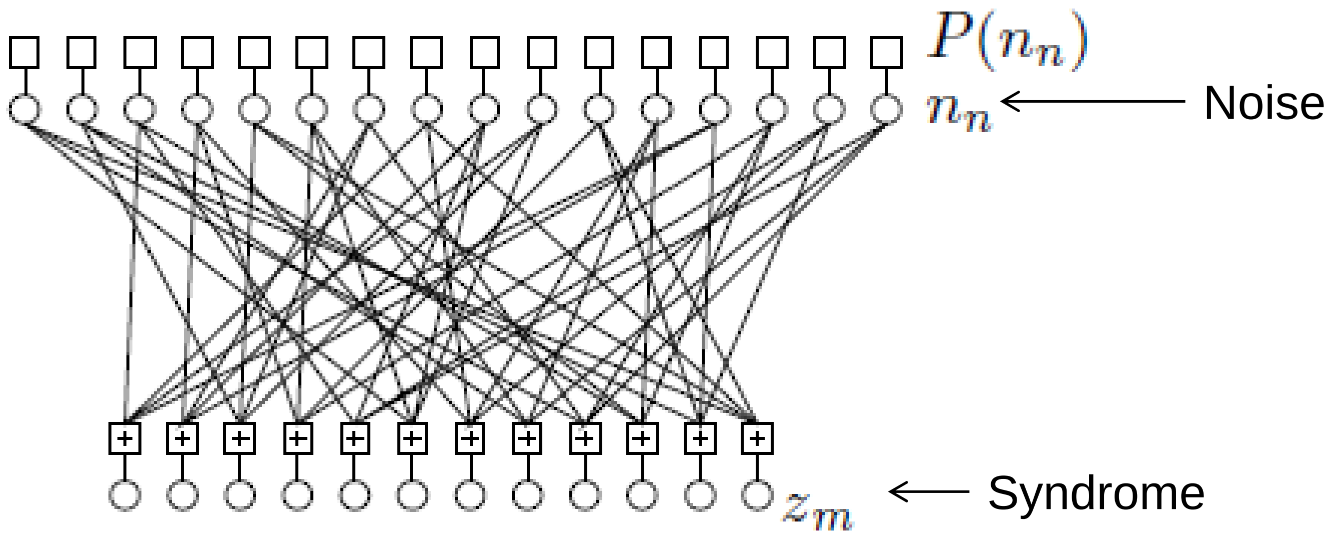
Error-free after 13 iterations

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

David MacKay
Information Theory, Inference, and Learning Algorithms

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



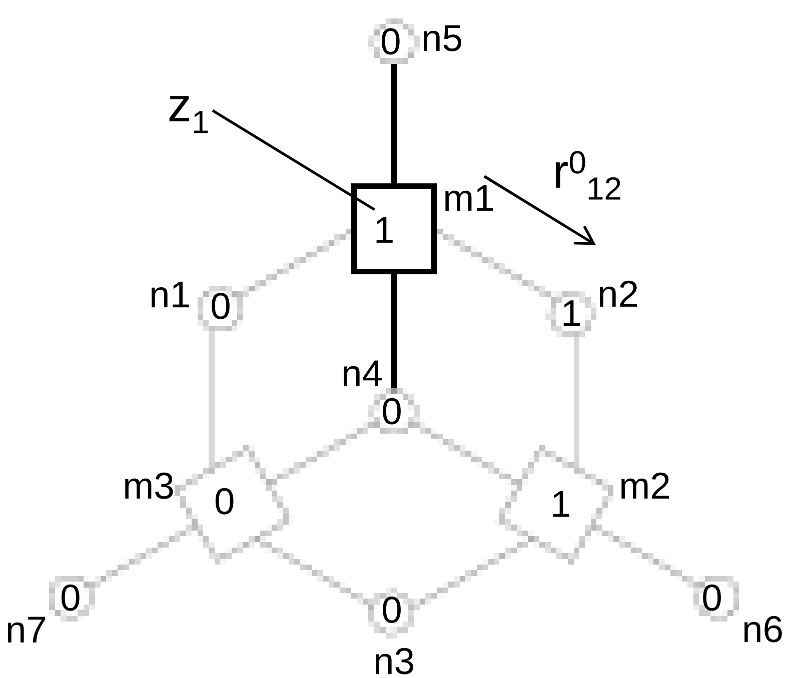
Iterative decoding --- Matlab example

End

How to decode

m indexes checks
n indexes bits

r^0_{12} 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: "-"

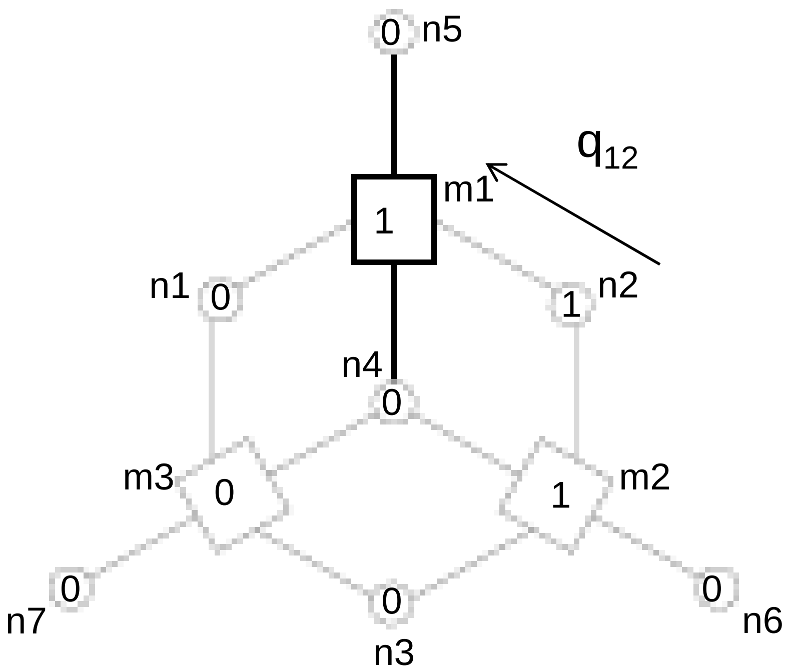
```
Hypothesized [list of hypotheses
  x            summed over in r⁰₁₂ calc]

0-000 → P(z1=1|x2=0,w=000) = 0
0-001 → P(z1=1|x2=0,w=001) = 1
0-010 → P(z1=1|x2=0,w=010) = 1
0-011 → P(z1=1|x2=0,w=011) = 0
1-000 → P(z1=1|x2=0,w=100) = 1
1-001 → P(z1=1|x2=0,w=101) = 0
1-010 → P(z1=1|x2=0,w=110) = 0
1-011 → P(z1=1|x2=0,w=111) = 1
```

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

m indexes checks
n indexes bits

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

Value of syndrome (directly determined by data!)

Hypothesized bit settings

How likely (approx.) is the bit vector being considered in outer loop?

$$r^{\boxed{0}}_{mn} = \sum_{\{x_{n'}:\, n' \in \mathcal{N}(m)\backslash n\}} P\left(z_m \mid x_n = \boxed{0}, \{x_{n'}:\, n' \in \mathcal{N}(m)\backslash n\}\right) \prod_{n' \in \mathcal{N}(m)\backslash n} q_{mn'}^{x_{n'}}$$

Outer loop

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

Inner loop

`r0(i,jp) = r0(i,jp) + PzGivenX0*qpp`

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) \ n means every bit associated with check m, EXCEPT for bit n

r_{mn}: 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

$$r_{mn}^{0} = \sum_{\{x_{n'}:\, n' \in \mathcal{N}(m)\backslash n\}} P\left(z_m \,|\, x_n = 0,\ \{x_{n'}:\ n' \in \mathcal{N}(m)\backslash n\}\right) \prod_{n' \in \mathcal{N}(m)\backslash n} q_{mn'}^{x_{n'}}$$

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

$$r_{mn}^{1} = \sum_{\{x_{n'}:\, n' \in \mathcal{N}(m)\backslash n\}} P\left(z_m \,|\, x_n = 1,\ \{x_{n'}:\ n' \in \mathcal{N}(m)\backslash n\}\right) \prod_{n' \in \mathcal{N}(m)\backslash n} q_{mn'}^{x_{n'}}.$$

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

$$q^0_{mn} = \alpha_{mn}\, p^0_n \prod_{m' \in \mathcal{M}(n)\setminus m} r^0_{m'n}$$

$$q^1_{mn} = \alpha_{mn}\, p^1_n \prod_{m' \in \mathcal{M}(n)\setminus m} r^1_{m'n}$$

$$q^0_n = \alpha_n\, p^0_n \prod_{m \in \mathcal{M}(n)} r^0_{mn},$$

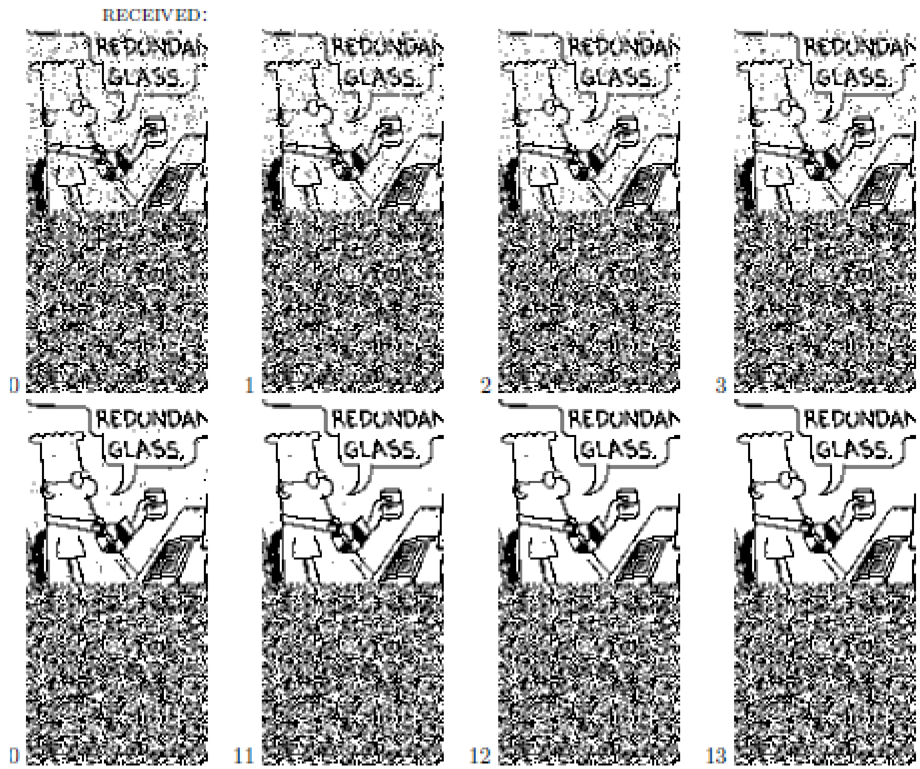
$$q^1_n = \alpha_n\, p^1_n \prod_{m \in \mathcal{M}(n)} r^1_{mn}.$$

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}

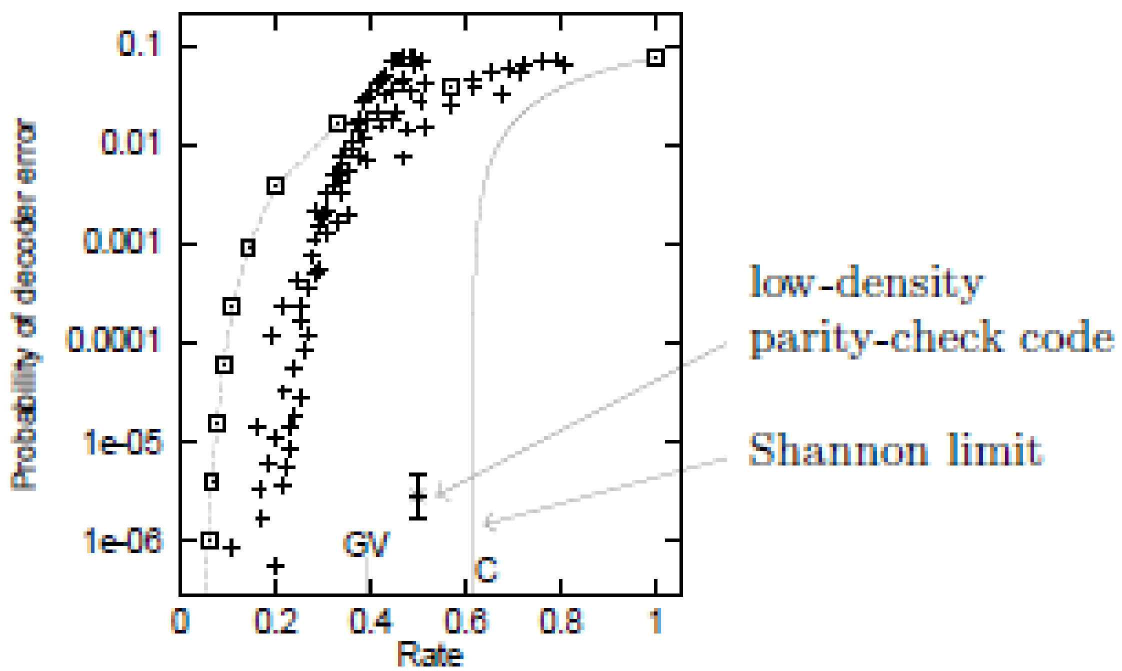
http://www.inference.phy.cam.ac.uk/mackay/codes/gifs/

DECODED:

Error-free after 13 iterations

David MacKay
Information Theory, Inference, and Learning Algorithms

Performance of LDPC codes

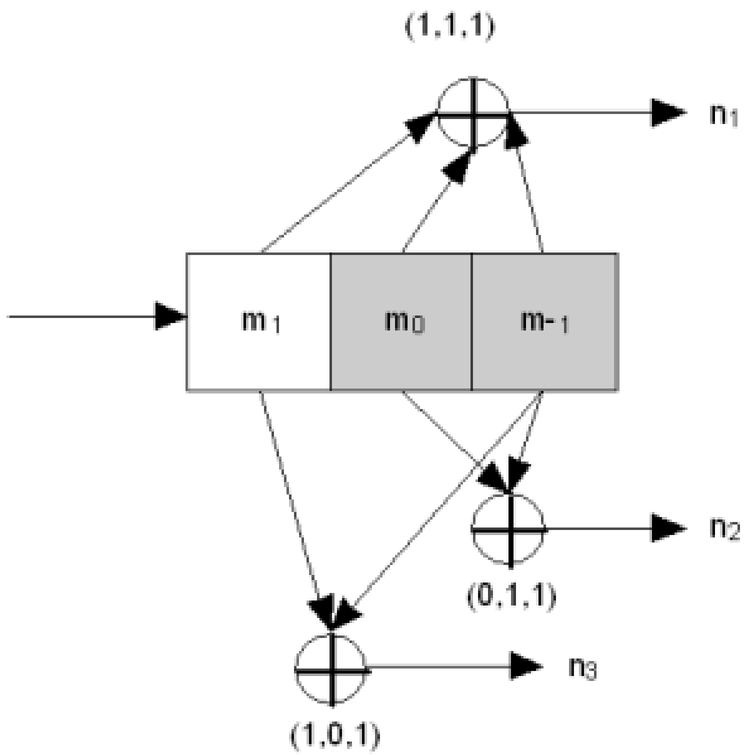


Fountain codes

- 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

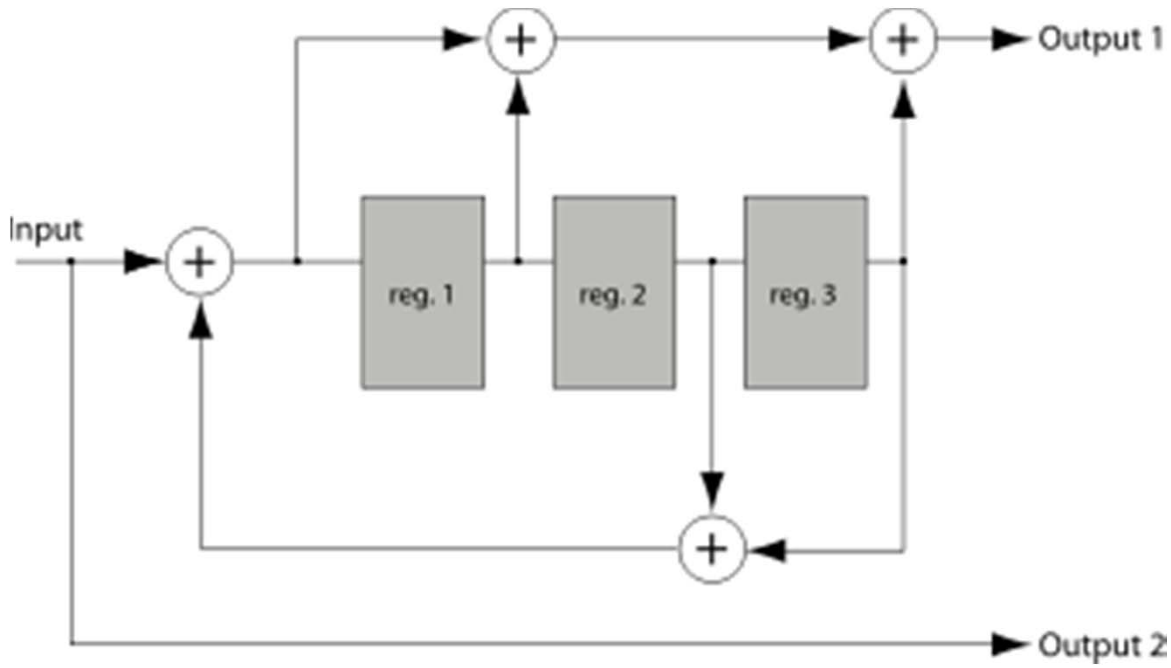
- Can use LFSR as encoder!



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