6.02 Spring 2012
Lecture #4

• Linear block codes - Properties
• Rectangular parity, Hamming

Idea: Embedding for Structural Separation

Encode so that the codewords are "far enough" from each other.
Likely error patterns shouldn't transform one codeword to another.

Code: nodes chosen in hypercube + mapping of message bits to nodes.

If we choose $2^k$ out of $2^n$ nodes, it means we can map all k-bit message strings in a space of n-bit codewords.
The code rate is $k/n$.

Linear Block Codes

Block code: $k$ message bits encoded to $n$ code bits i.e., each of $2^k$ messages encoded into a unique n-bit combination via a linear transformation.
Set of parity equations (in GF(2)) represents code.

Key property: Sum of any two codewords is also a codeword $\Rightarrow$ necessary and sufficient for code to be linear.

$(n,k,d)$ code has rate $k/n$.
Sometime written as $(n,k,d)$, where $d$ is the Hamming Distance of the code.
Examples: What are n, k, d here?

\{000, 111\} \quad (3,1,3). Rate= 1/3.
\{0000, 1110, 0011, 1111\} \quad (4,2,2). Rate = 1/2.
\{00000\} \quad (5,0,2). Rate = 0!
\{1111, 0000, 0001\}
\{1111, 0000, 0010, 1100\}
\{000000, 110001, 110110, 000111\}
\{010101, 010111, 100110, 010101\}
\{111100, 001100, 001111, 111111\}
\{7,4,3\} code. Rate = 4/7.

\begin{itemize}
  \item Split data into k-bit blocks
  \item Add (n-k) parity bits to each block using (n-k) linear equations, making each block n bits long
\end{itemize}

The entire block is called the "code word in systematic form"

Example: Rectangular Parity Codes

<table>
<thead>
<tr>
<th>D_1</th>
<th>D_2</th>
<th>P_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_3</td>
<td>D_4</td>
<td>P_2</td>
</tr>
<tr>
<td>P_3</td>
<td>P_4</td>
<td></td>
</tr>
</tbody>
</table>

P_1 is parity bit for row #1
P_2 is parity bit for column #2

-Idea: start with rectangular array of data bits, add parity checks for each row and column. Single-bit error in data will show up as parity errors in a particular row and column, pinpointing the bit that has the error.

\begin{itemize}
  \item Parity for each row and column is correct ⇒ no errors
  \item Parity check fails for row #2 and column #2 ⇒ bit D_4 is incorrect
  \item Parity check only fails for row #2 ⇒ bit P_2 is incorrect
\end{itemize}

(n,k) Systematic Linear Block Codes

- Every linear code can be represented in systematic form

Rectangular Code Corrects Single Errors

Claim: The HD of the rectangular code with r rows and c columns is 3. Hence, it is a single error correction (SEC) code.

Code rate = rc / (rc + r + c).

If we add an overall parity bit P, we get a (rc+r+c+1, rc, 4) code

Improves error detection but not correction capability

Proof: Three cases.

1. Msgs with HD = 1 ⇒ differ in 1 row and 1 col parity
2. Msgs with HD = 2 ⇒ differ in either row OR col or both ⇒ HD = 4 here.
3. Msgs with HD = 3 or more ⇒ systematic code so differ in that many bits
Decoding Rectangular Parity Codes

Receiver gets possibly corrupted word, $w$.
Calculates all the parity bits from the data bits.
If no parity errors, return $rc$ bits of data.
Single row or column parity bit error $\rightarrow$ $rc$ data bits are fine, return them
If parity of row $x$ and parity of column $y$ are in error, then the data bit in the $(x,y)$ position is wrong; flip it and return the $rc$ data bits
All other parity errors are uncorrectable. Return the data as-is, flag an “uncorrectable error”

Let’s do some rectangular parity decoding

Received codewords

1. Decoded message bits: ____________

2. Decoded message bits: ____________

3. Decoded message bits: ____________

Towards More Efficient Codes: (7,4,3) Hamming Code Example

- Use multiple parity bits, each covering a subset of the data bits.
- No two message bits belong to exactly the same subsets, so a single-bit error will generate a unique set of parity check errors.

How Many Parity Bits Do We Really Need?

- We have $n-k$ parity bits, which collectively can represent $2^{n-k}$ possibilities
- For single-bit error correction, parity bits need to represent two sets of cases:
  - Case 1: No error has occurred (1 possibility)
  - Case 2: Exactly one of the code word bits has an error ($n$ possibilities, not $k$)
- So we need $n+1 \leq 2^{n-k}$
- Rectangular codes do not satisfy the equality
- Hamming codes correct single errors with this minimum number of parity bits
  $(7,4,3), \ldots, (2^m - 1, 2^m - 1 - m, 3)$
Logic Behind Hamming Code Construction

- Idea: Use parity bits to cover each axis of the binary vector space
  - That way, all message bits will be covered with a unique combination of parity bits

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary index</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>(7,4) code</td>
<td>P₁</td>
<td>P₂</td>
<td>D₁</td>
<td>P₃</td>
<td>D₂</td>
<td>D₃</td>
<td>D₄</td>
</tr>
</tbody>
</table>

P₁ with binary index 001 covers

P₂ = D₁ + D₂ + D₄
P₃ = D₁ + D₃ + D₄

D₁ with binary index 011
D₂ with binary index 101
D₃ with binary index 111
D₄ with binary index 111