Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
Department of Mechanical Engineering

6.050J/2.110J **Information and Entropy** **Spring 2006**

Issued: February 23, 2006 **Due: March 3, 2006**

Problem Set 4

Problem 1: Nostalgia from 6.050!

Little-Nibble.com is a startup company that deals with small chunks of data. Alyssa P. Hacker, an MIT alumna who is now with Little-Nibble.com, remembers that as a freshman who had lots of enthusiasm in 6.050, she once devised a code that would be appropriate in this application. The $(5, 2, 3)$ code was for 2-bit chunks of data ($k = 2$) that used 5-bit code words ($n = 5$) and permitted single-error correction because the minimum Hamming distance is 3 ($d = 3$). The first two bits of each code-word, she recalls, were the data bits being coded; the other three were parity bits. Unfortunately, when she finds the code among her old notes, part of the codebook (the part shown with question marks below) cannot be read due to aging!

Input		Codebook				
0	0	0	0	0	0	0
0	1	0	1	?	?	?
1	0	1	0	?	?	?
1	1	1	1	?	?	?

Table 4–1: Alyssa's Codebook, as transformed during the years

a. There are many ways to implement a code of this sort. Find one of them and complete the codebook showing the codes for each of the four input strings.

b. Of the 32 possible bit strings the decoder might encounter, how many are legal codes?

c. Some of these 32 have Hamming distance one from a legal code and may therefore be corrected to the nearest legal value under the assumption they were produced by a single error. How many?

d. Still others have a Hamming distance greater than one from any legal code, and therefore can only be produced by multiple errors. How many?

Problem 2: Divide to encode ...

There are two basic techniques for error correction in communication channels. In one, enough additional bits are added to each block to allow the recipient to tell whether an error has happened, and if so, which bit(s) are affected. This technique is sometimes called "Forward Error Correction." The Hamming code is a good example. The other technique is possible if the recipient has some way to request retransmission of a block, for example if the channel can carry messges in both directions, or if another auxiliary back- channel is available. In this case, all the recipient needs to know is whether an error has happened, i.e., error detection is needed but not error correction. An example is TCP/IP.

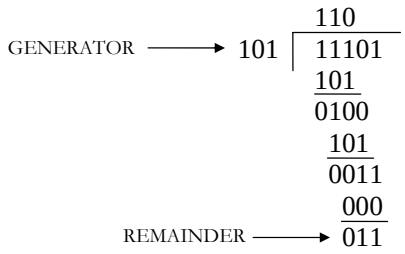
In this problem we will work with a simple channel encoding useful for error detection. The codes are known as "cyclic codes." The basic idea is very simple. The original blocks are binary strings of length k. The codewords are binary strings of length n (which is greater than k, of course). Each codeword can be

thought of as representing a nonnegative integer using the standard binary code, with most significant bits first.

In cyclic codes, the valid codewords are those which are a multiple of some number called the "generator." Thus if the generator is 5, only 20 % of the codewords are valid, and the others represent errors. The art of designing a high quality cyclic code lies in the choice of generator and the mapping of each block into a codeword. It is easy to do it wrong; for example, if the generator is 4, or 100 in binary, then the valid codewords all end with 00 and an error that changes the codeword 1100 to 0100 would not be detected. In cyclic codes, the various arithmetic operations are all done with binary strings modulo 2, and hence subtraction is done with XOR operations.

It is easy to detect whether a codeword received after transmission through the channel is valid. You have to divide it by the generator and see if there is any residue. In binary arithmetic, this can be done with a technique like long division. (You do remember long division from your high-school days, don't you?) You "subtract" either the generator or a string of zeros at each stage, depending on whether the remainder of the previous subtraction starts with 1 or 0. If there is a nonzero remainder after the last such subtraction, the original number is not divisible by the generator.

For example, to divide the codeword 11101 by the generator 101:



We see that since there is a remainder 011, the codeword is not valid, and hence an error must have happened. We cannot tell which bit in the codeword might have been corrupted.

a. For the generator $G = 10011$, detect possible errors of the received messages 1100000000 and 1110100010.

Now we need to find a way to generate the codewords. Suppose the generator G is represented in $r + 1$ binary digits starting with 1. The blocks to be encoded have k bits and the codewords more than k bits.

b. Your roommate suggests adding r zeros to the end of the block, so $n = k + r$. If the result is not divisible by G (modulo 2), either add or subtract the remainder from the last r bits. Apply this technique to encode the messages $M_1 = 110000$ and $M_2 = 111010$ for the generator $G = 10011$.

c. You wonder whether this scheme results in codewords for which every possible single error can be detected. Let E denote the string of length n with 1 in each position in which an error has happened. Thus for single errors, the Hamming distance between the transmitted and received codewords is 1, and there is exactly one 1 in E. Thus there are exactly n such error strings E. Give a simple rule for telling which of these n error strings are detected.

d. Show that we can detect all single-bit errors using this method if G has two non-zero digits

e. **(Extra credit)** Try to generalize this result and find more classes of errors that can be detected with appropriate choices of G.

Laboratory Experiment

This experiment will take you into the world of linear codes used to protect transmitted data from errors caused by noise in the environment. Such codes are actively used in space transmissions, cellular phones, and CD players. These codes are applied to blocks of k bits of a message to yield n bits to be transmitted, with a minimum Hamming distance between legal codes of d. We refer to these codes as (n, k, d) linear codes or sometimes simply as (n, k) codes.

The encoder receives k bits to be transmitted and adds $n-k$ bits of parity defined by rules expressed in a generator matrix G. The block of n bits is then sent through some medium, e.g. air, floppy disk, etc., where it may encounter noise which produces errors. When the decoder receives the block of n bits it strips off the $n-k$ bits of parity leaving the k bits of the message. But before discarding the parity bits, the decoder uses them to correct any errors it finds. The number of errors that can be detected or corrected depends on the linear codes used to generate the parity bits. The decoder uses a parity-check matrix H.

Consider the $(7, 4, 3)$ Hamming code: 4 bits for the message and 3 bits for the parity codes totaling up to 7 bits. The following is one possible generator matrix G and the corresponding parity check matrix H.

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{4–1}$$

	Message				Parity		
Bits	1	2	3	4	5	6	7
Parity 1				x	x	x	x
Parity 2		x	x			x	x
Parity 3	x		x		x		x

Table 4–1: Parity

Table 4–1 is based on the matrix H but labels the first four bits as message and the rest as parity.

These codes are called linear codes because the operations are done by matrix multiplication (a special form that uses XOR operations in place of normal arithmetic multiplication and addition). Let's multiply the matrix G by the four-bit message A to produce a seven-bit codeword:

```
>> A = [1 1 0 1];
>> B = A*G
B =
1 1 0 1 2 2 3
```

Oops. Note that this is not what we want for the 7-bit codeword. MATLAB did conventional matrix multiplication. We have to change all even values to 0 and odd values to 1 to get what we want, `[1 1 0 1 0 0 1]`. This can be done several ways, for example by using mod-2 remainders

```
>> B = mod(B, 2)
B =
1 1 0 1 0 0 1
```

Notice that in B the first 4 bits are A and the last 3 bits are the parity. This is because our generator G matrix has this property (the first four columns of G form an identity matrix). There are other generator matrices that still yield a $(7, 4, 3)$ Hamming code but place the bits of the original message in other positions.

The matrix H is used by the decoder to discover any errors:

```
>> C = mod(B*H', 2)
C =
0 0 0
```

where the apostrophe is the symbol reserved by MATLAB for matrix transpose; in mathematical formulae we use a superscript T to indicate matrix transpose (the matrix formed by flipping a matrix about its main diagonal – H is a 3×7 matrix and H^T is a 7×3 matrix). The fact that C is entirely 0 means there were no errors to correct, so the decoder simply strips off the parity bits. Now let us assume that an error has occurred in bit position 3 so the decoder receives the message as `[1 1 1 1 0 0 1]` rather than `[1 1 0 1 0 0 1]`. Now,

```
>> B(3) = ~B(3);
>> C = mod(B*H', 2)
C =
0 1 1
```

Due to our choice of H, the result `[0 1 1]` tells us directly the position of the error: 011 translates to decimal 3; the bit position of the error.

Use MATLAB to reproduce these results and try one or two other messages. Then you will be ready for Problem 3. Do not include this lab experiment as part of what you turn in.

Hint: you can define a matrix like G by something like `G = [1 0 0 0 0 1 1; 0 1 0 0 1 0 1;...]`

Problem 3: A Different Hamming Code

There are many possible Hamming code generator matrices G and corresponding decode matrices H. The example in the laboratory experiment above had the useful property that the binary representation of the location of the error is the result of using the H matrix. This property was a result of the fact that H was made of columns with these binary representations in order.

In a more general case, (see Error Correction with Hamming Codes) the bit pattern produced by multiplying the received code word by H can be compared with the columns of H, and the location of the fault inferred by which column if any is matched. For this problem use these alternate matrices (different from the matrices used in the Laboratory Experiment above):

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4–1}$$

Hint: It is easiest to do this problem using MATLAB, but if you prefer you can use paper and pencil. If you do use MATLAB, put your code in the file `ps3p2.m` and all your comments in your diary named `ps3diary`.

a. Generate a test pattern PATTERN which consists of all possible four-bit messages. Make this in the form of a matrix with four columns and 16 rows.

b. Generate the "codebook" which is a matrix containing in each row the seven-bit codeword from the corresponding row of PATTERN. You can get this with a single matrix multiplication followed by the mod operation. Call this matrix CODEBOOK.

c. How many errors can the decoder using the matrix H correct? How many errors can it detect if it does no corrections?

d. Verify that this $(7, 4, 3)$ Hamming code works by using it to transmit and receive 3 messages of your choice, without errors. In this context, "transmit and receive" means select a four-bit word, find its codebook entry by using the matrix G, check it for errors using the matrix H, and if necessary make corrections.

e. Repeat with the same messages but this time introduce one error in each message and verify that the use of the matrix H reveals the location of the error (remember that to identify which bit is at fault you compare with the columns of H). In one of these cases make the error happen in a parity bit.

f. Using one of these messages, introduce two errors and see what the decoder does. Compare the original message with the "corrected" one.

g. The three bits obtained with the check matrix H are often termed syndrome. The syndromes indicate which bit has been flipped (assuming one and only one error has occurred). Given our choice of H, construct a table that associates each syndrome with the bit it indicates. And compare your result to the matrix H and the last 3 columns of G, how are they all related?

Turning in Your Solutions

If you used MATLAB, you should have M-files and a diary. You may turn in this problem set by e-mailing your M-files and diary to 6.050-submit@mit.edu. Do this either by attaching them to the e-mail as text files, or by pasting their content directly into the body of the e-mail (if you do the latter, please indicate clearly where each file begins and ends). Alternatively, you may turn in your solutions on paper in room 38-344. The deadline for submission is the same no matter which option you choose.

Your solutions are due 5:00 PM on Friday, March 3, 2006. Later that day, solutions will be posted on the course website.