COS 330: Great Ideas in Theoretical Computer Science

Fall 2025

Precept 10

My perfect code

Learn

In lecture we learned about error-correcting codes and how they allow us to reliably transmit or store information. Here we will review some of the key definitions, and we will also see another family of codes.

Let's recall the foundational definitions from class. Suppose we have n-bit messages we want to store in a potentially corruptible storage device or that we want to transmit over a noisy channel. We *encode* each n-bit message into a longer m-bit string (with $m > n$), adding redundancy, and then *decode* it back after transmitting it or reading it from our storage device.

Formally, a *binary code* consists of an encoding function $\mathrm{Enc} : \{0,1\}^n \to \{0,1\}^m$ and a decoding function $\mathrm{Dec} : \{0,1\}^m \to \{0,1\}^n$. The encoding is injective (different messages get different encodings), so we can equivalently think of a code through its *codewords*: the set $C = \{\mathrm{Enc}(x) : x \in \{0,1\}^n\} \subseteq \{0,1\}^m$ of all possible m-bit encodings. We say the code has *length* m, *dimension* n (the number of message bits), and contains $|C| = 2^n$ codewords.

The *Hamming distance* $\mathrm{dist}(y, y')$ between two binary strings $y, y' \in \{0,1\}^m$ is the number of positions where they differ. The *minimum distance* of a code C is:

$$d_{\min}(C) = \min_{c,c' \in C, c \neq c'} \mathrm{dist}(c, c')$$

Why does minimum distance matter? If $d_{\min}(C) \geq 2e + 1$, then we can *correct* up to e errors, and if $d_{\min}(C) \geq d + 1$, we can *detect* up to d errors. For correction, the intuition is that if we transmit codeword c but receive a corrupted word y (where at most e bits were flipped), then c is the unique codeword within distance e of y, so the decoder can find c by looking for the closest codeword. For detection, even if d bits are flipped, the result cannot be another codeword (since codewords are at least distance $d + 1$ apart), so we know an error occurred.

The above gives us two notions of efficiency in codes. First, *distance*: we want $d_{\min}(C)$ to be large so we can correct or detect many errors. Second, *redundancy*: we want the length m to be as close to the message length n as possible, minimizing the extra bits we add. These goals are in tension—more redundancy typically gives higher distance—so good codes balance both concerns.

Hamming Codes

Now let's turn to constructing good codes. Let's start with a simple idea: *parity checks*.

Suppose we want to send a 4-bit message. We could add a single parity bit that equals the XOR (sum mod 2) of all 4 message bits. For example, to send $(1, 0, 1, 1)$, we'd compute the parity

$1 \oplus 0 \oplus 1 \oplus 1 = 1$ and send $(1, 0, 1, 1, 1)$. This gives us a code with minimum distance 2: any single-bit error changes the parity, so we can *detect* one error. But we can't *correct* it—we know an error happened, but not where.

To correct errors, we need minimum distance 3. Here's the key insight: instead of one parity check covering all bits, use *multiple overlapping parity checks*. If different parity checks fail, the pattern of failures tells us exactly which bit was flipped!

Hamming codes are a family of codes that implement this idea systematically. For any integer $r \geq 2$, we'll construct a Hamming code with length $m = 2^r - 1$ and dimension $n = 2^r - 1 - r$.

Encoding. To encode an n-bit message, we create an m-bit codeword as follows: the first n bits are the message bits (unchanged), and we append r *parity check bits* at the end:

$$c = (c_1, \ldots, c_m) = (x_1, x_2, \ldots, x_n, p_0, p_1, \ldots, p_{r-1}),$$

where c_j denotes the bit in position j, and p_i is the i-th parity bit, sitting at position $n + 1 + i$ (so p_0 is at position $n + 1$, p_1 at $n + 2$, and so on). The exact positions of the parity bits are not important for the distance proof - we just fix this convention since it helps with clarity. (If you are not convinced with this, try showing that if you permute the bits of all codewords of any code in the same way, the minimum distance remains unchanged.)

The key idea is to view each position $j \in \{1, 2, \ldots, m\}$ via its r-bit binary representation (with leading zeros as needed). For example, with $r = 3$ (so $m = 7$), the positions are:

$$1 = 001,\ 2 = 010,\ 3 = 011,\ 4 = 100,\ 5 = 101,\ 6 = 110,\ 7 = 111.$$

For each $i \in \{0, 1, \ldots, r - 1\}$ we define a parity *check* as follows: look at all positions j whose binary representation has a 1 in bit i (counting bits from the right, starting at 0), and require that the XOR of the bits in those positions is 0 (even parity). Formally, for each i we require

$$\bigoplus_{j:\text{bit } i \text{ of } j \text{ is } 1} c_j = 0. \qquad (*_i)$$

In words, for $r = 3$ these checks are:

- Check 0 (bit 0): positions $1, 3, 5, 7$ (all indices with least significant bit 1).
- Check 1 (bit 1): positions $2, 3, 6, 7$.
- Check 2 (bit 2): positions $4, 5, 6, 7$.

Note that these sets include the parity positions themselves (e.g., position 5 appears in checks 0 and 2).

The r parity bits $p_0, \ldots, p_{r-1}$ are chosen so that all r equations $(*_0), \ldots, (*_{r-1})$ hold. Given the message bits $x_1, \ldots, x_n$, this system of r XOR equations has a unique solution for $p_0, \ldots, p_{r-1}$, so the encoding is well-defined. (This isn't obvious, but can be checked with a bit of linear algebra. If you want some intuition, there are r equations with r unknowns, and the equations are linearly independent.)

We do not need a closed-form formula for p_i in general, but for concreteness here is the case $r = 3$, $n = 4$ (so $m = 7$). Solving the three parity equations $(*_0), (*_1), (*_2)$ gives

$$C(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4,\ x_2 \oplus x_3 \oplus x_4,\ x_1 \oplus x_3 \oplus x_4,\ x_1 \oplus x_2 \oplus x_4).$$

You can check, for example, that in this codeword

$$c_1 \oplus c_3 \oplus c_5 \oplus c_7 = 0, \quad c_2 \oplus c_3 \oplus c_6 \oplus c_7 = 0, \quad c_4 \oplus c_5 \oplus c_6 \oplus c_7 = 0,$$

exactly matching the three checks above.

Distance. We'll show that any two distinct codewords differ in at least 3 positions.

Take any two distinct codewords

$$c^{(1)} = (c_1^{(1)}, \ldots, c_m^{(1)}) \quad \text{and} \quad c^{(2)} = (c_1^{(2)}, \ldots, c_m^{(2)}).$$

Let

$$S = \{\, j \in \{1, \ldots, m\} : c_j^{(1)} \neq c_j^{(2)} \,\}$$

be the set of all positions where they differ. The Hamming distance between the two codewords is exactly $|S|$, so it suffices to show that $|S| \geq 3$.

Both codewords satisfy all r parity checks $(*_i)$. Fix some $i \in \{0, \ldots, r-1\}$. For each codeword $t \in \{1, 2\}$ we have

$$\bigoplus_{j\text{:bit } i \text{ of } j \text{ is } 1} c_j^{(t)} = 0.$$

XOR-ing these two equations gives

$$\bigoplus_{j\text{:bit } i \text{ of } j \text{ is } 1} \left(c_j^{(1)} \oplus c_j^{(2)}\right) = 0.$$

The value $c_j^{(1)} \oplus c_j^{(2)}$ is 1 exactly when $j \in S$ and 0 otherwise. Thus this says:

For each i, among the positions in S whose i-th binary bit is 1, there are an even number of them.

Now suppose, for contradiction, that $|S| \leq 2$.

Case 1: $|S| = 1$. Say $S = \{\ell\}$. Look at any bit position i where the r-bit binary representation of ℓ has a 1. For this i, exactly one position in S has bit $i = 1$, so the number of positions in S with i-th bit 1 is 1, which is odd. This contradicts the condition above. Hence $|S| \neq 1$.

Case 2: $|S| = 2$. Say $S = \{\ell_1, \ell_2\}$ with $\ell_1 \neq \ell_2$. Because the indices ℓ_1 and ℓ_2 are different, their r-bit binary representations differ in at least one bit position. Let i be such that the i-th bit of ℓ_1 and ℓ_2 are different. Then exactly one of ℓ_1, ℓ_2 has bit $i = 1$, so among S there is an odd number (namely 1) of positions with i-th bit equal to 1. Again this contradicts the condition above. Hence $|S| \neq 2$.

Therefore, we cannot have $|S| = 1$ or $|S| = 2$, so necessarily $|S| \geq 3$. That is, any two distinct codewords differ in at least 3 positions, so the Hamming code has minimum distance 3.

In particular, this means the code can detect up to 2 bit errors and *correct* any single-bit error. We won't go over how to efficiently encode or decode messages using Hamming codes, but there are well-known algorithms running in linear time in n to do so.

Practice

Problem 1: The Hamming Bound and Perfect Codes

In this problem, we'll explore lower bounds on the redundancy of error-correcting codes, culminating in a proof of why Hamming codes are called "perfect" codes.

Let $C \subseteq \{0,1\}^m$ be a binary code with minimum distance $d_{\min}(C) = 3$ (note that this is any code, not necessarily a Hamming code.) As we saw in the Learn section, this means C can correct up to 1 error.

For any codeword $c \in C$, define the *Hamming ball of radius 1* centered at c as:

$$B_1(c) = \{x \in \{0,1\}^m : \text{dist}(x, c) \leq 1\}$$

This is the set of all binary strings within distance 1 of c.

(a) Show that $|B_1(c)| = 1 + m$ for any $c \in \{0,1\}^m$.

(b) Prove that if C has minimum distance 3, then the balls $B_1(c)$ are pairwise disjoint for distinct codewords $c, c' \in C$.

Hint: Use the triangle inequality for Hamming distance: $\text{dist}(x, c') \geq \text{dist}(c, c') - \text{dist}(x, c)$.

(c) Use parts (a) and (b) to prove the *Hamming bound*: if C has $|C|$ codewords, length m, and minimum distance 3, then:

$$|C| \cdot (1 + m) \leq 2^m$$

or equivalently, $|C| \leq \frac{2^m}{1+m}$.

Hint: The balls are disjoint subsets of $\{0,1\}^m$.

(d) A code is called *perfect* if it achieves equality in the Hamming bound. Show that the Hamming code with parameters $m = 2^r - 1$ (length) and $n = 2^r - 1 - r$ (dimension) is perfect.

Hint: Verify that $|C| \cdot (1 + m) = 2^m$ by substituting the values, recalling that $|C| = 2^n$.