Class 6 Exercises

CS250/EE387, Winter 2025

1. **(QR Codes I)** This 21 × 21 QR code is encoding a message with a fair amount of redundancy (level "Q" in QR-code lingo).

More precisely[1], the encoding process starts with a message (in this case, plain text) and interprets it as $k = 13$ elements of $\mathbb{F}_{256}$ (aka, as 13 bytes, where each byte encodes a unicode symbol). Then it encodes these k field elements using a RS code over $\mathbb{F}_{256}$ with $k = 13$ and $n = 26$. It takes the 26 codeword symbols (elements of $\mathbb{F}_{256}$), and treats them as bytes again. Then these bytes get written into the QR code in a specified order: the 8 bits that represent a given symbol in $\mathbb{F}_{256}$ get mapped to contiguous 4×2 blocks. A mask is applied over the top, and the remaining pixels are used for formatting information (what level of encoding, what type of message, those little squares to help your phone position it, what the mask is, etc).

   (a) What is the rate of the (256-ary) RS code used? What is the distance?

   (b) How many pixels in the QR code are auxiliary formatting information?

   (c) Explain how you can view this code (not including the auxiliary pixels or the mask) as a binary concatenated code. What is the rate of this binary code? What can you say easily[2] about the distance? (Bonus: can you figure out the actual distance?)

Solution

(a) The rate of the original RS code is 1/2, and the distance is 14.

(b) The number of pixels used for auxiliary info is 233 out of 441; this is because there are 26 bytes, or $8 \cdot 26$ bits, used for the actual codeword, and $21^2 - 8 \cdot 26 = 223$.

(c) We can view the QR code as a concatenated code where the outer code is RS and the inner code is the trivial code which maps $x \in \mathbb{F}_{256}$ to the corresponding 8-bit vector. The rate of this concatenated code is still 1/2, since the inner code has rate 1, the outer code has rate 1/2, and the rates multiply. The distance is at least $14 \cdot 1 = 14$, which puts the relative distance at $\delta = 14/(26 \cdot 8) = 0.067$. (I don't know what the actual distance is.)

[1] This is my understanding of how QR codes work; take it as fact for this problem, but if you are inspired to read up on it and happen to be able to correct me, please let me know!

[2] We're looking for a statement like "The distance is at least x." The statement "the distance is at least 0" is too easy.

2. **(QR Codes II)** Building on the previous problem, let's do a thought experiment about a different way a QR code could work, based on concatenated codes.

   (a) Show that there is a binary linear code of dimension 7, length 8, and distance 2.

   Now, back to the QR codes, instead of choosing an RS code over $\mathbb{F}_{256}$, let $\mathcal{C}_{out}$ be an RS code with $k = 15$ and $n = 26$ over $\mathbb{F}_{128}$. Let $\mathcal{C}_{in}$ be a binary code from part (a). Now encode a QR code as outlined in the previous problem, but instead with the concatenated code $\mathcal{C} = \mathcal{C}_{out} \circ \mathcal{C}_{in}$.

   (b) What is the rate and designed distance of $\mathcal{C}$? How does it compare with the one in the previous problem?

   (c) Why do you think that QR codes don't actually work this way?

Solution

(a) Consider the parity-check code of length 8:

$$\mathcal{C}_{in} = \left\{ (x_1, \ldots, x_8) \,:\, \sum_i x_i = 0 \mod 2 \right\}.$$

The lowest-weight nonzero codeword in this code has weight 2, so the distance is 2. The code is defined by a single linear relationship, so the dimension is $8 - 1 = 7$.

(b) The rate is

$$R = \frac{15}{26} \cdot \frac{7}{8} = \frac{105}{208},$$

which in particular is slightly larger than $1/2$. The designed distance is

$$d \geq 12 \cdot 2 = 24,$$

which is larger than 14. Thus, at least on paper and only considering the designed distance, this code gets a better trade-off than the one used in a QR code.

(c) There are several good reasons.

- First, I'm not actually sure what the actual distances of these codes are, perhaps the first one isn't so bad.
- Second, think about the sorts of errors that a QR code expects. It's not so much that individual pixels all over the place are going to get messed up. Rather, whole blocks of pixels might be occluded or drawn over. So it actually makes sense to think of this not as a concatenated code, but as a code over $\mathbb{F}_{256}$, and we hope that when things get messed up, it's whole bytes at a time. In this case, the RS code over the larger alphabet is the better choice.
- The scheme in the previous problem is easier to describe and implement.
- While the rate of this code in this problem is better, the message length is 105, which is not a multiple of 8. This is fine if I'd like to encode 105 bits, but if my message is, say, a phrase in unicode, the message length is going to be a multiple of 8.

3. **(An application of BCH codes)** Consider the following problem (which doesn't have to do with coding theory).

   - A sender (Sally) sends a set $S \subseteq \{0,1\}^m$ of size N to a receiver (Rolanda). Think of the elements of S as being packet headers or something; Sally is going to send N packets to Rolanda, with packet headers in S. (Also, think of N as being pretty large, like $(2^m)/10$ or something like that.)

- Unfortunately, some of the packets get lost on the way, and Rolanda receives a set $T \subseteq S$. Let's say that $|T| = N - x$, so x packets have gone missing. (Think of $x \ll N$ as being WAY smaller than N).
- Rolanda would like to tell Sally which packets were lost, so that Sally can re-send them. The goal is to minimize the amount of communication from Rolanda back to Sally. Assume that Rolanda has a perfect communication channel back to Sally.
- (Note: let's assume that each packet is really really big (much bigger than the m-bit header), so Sally really doesn't want to re-send any packet she doesn't need to; in particular "resending all N packets" is not a good solution).

(a) Here are three simple schemes to use as a baseline.

i. Rolanda sends all the headers in T back to Sally. Sally computes $S \setminus T$ and resends the missing packets.

ii. Rolanda sends a binary vector $\mathbf{1}_T \in \{0,1\}^{2^m}$ back to Sally, to indicate which packets she received. Sally computes $S \setminus T$ and resends the missing packets.

How much communication from Rolanda back to Sally, measured in bits, is required in these simple schemes? If $N \approx (2^m)/10$ and $x = O(1)$, what is this in terms of N, in big-Oh notation?

Solution

The first scheme requires $(N - x)m$ bits. If $N \approx (2^m)/10$ and $x = O(1)$, this is $\Theta(N \log N)$ bits. The second is a bit better when x is so small, it requires only $N = 2^m$ bits.

(b) Now we'll work out a better scheme that uses BCH codes. Here's the outline of the scheme:

- Let's assume that the all-zero string will never be in S. (Why is this an okay assumption?)
- Sally sends $S \subseteq \{0,1\}^m$ to Rolanda, who receives $T \subset S$, as above.
- Rolanda computes $H\mathbf{1}_T$, where $\mathbf{1}_T \in \{0,1\}^{2^m-1}$ is the indicator vector for the set $T \subseteq \{0,1\}^m \setminus \{\vec{0}\}$, and where H is the parity-check matrix for an appropriate BCH code (you will need to choose the parameters!)
- Sally computes $v = H\mathbf{1}_S - H\mathbf{1}_T$.
- Sally (somehow...) recovers the identities of the missing packets from v, and sends them to Rolanda.

Answer the following:

i. Fill in the details. What are the parameters of the BCH code you should choose? How does Sally figure out which packets Rolanda needs? Why is it okay to ignore the all-zero string (up to, say, one extra bit of communication)? And what is the final amount of communication from Rolanda back to Sally?

Solution

First, it's fine to ignore the all-zero string since we can say that Rolanda can send back one additional bit indicating whether or not she got the all-zero string.
Next, notice that $v = H(\mathbf{1}_S - \mathbf{1}_T) = H \cdot \mathbf{1}_{S \setminus T}$. Note that $\mathbf{1}_{S \setminus T}$ is the indicator vector for the x packets that Rolanda is missing. Thus, Sally can figure out what these packets are by finding $S \setminus T$ given $H\mathbf{1}_{S \setminus T}$; this is exactly the syndrome decoding problem for BCH codes from x errors.
Thus, we need to choose the BCH code so that (a) it has length $n = 2^m - 1$; and (b) it can correct x errors. Fortunately we saw a code that can do this! $BCH(n, d)$ has length $n = 2^m - 1$ and has distance d, so we set $d = 2x + 1$.
The total amount of communication for the BCH part of the scheme is the number of

rows in the parity-check matrix H, which is

$$n - \dim(BCH(n,d)) = n - (n - \frac{d-1}{2}\log_2(n+1)) = \frac{d-1}{2}\log_2(n+1) = x \cdot m.$$

Then we add 1 for the zero header, so we get $xm + 1$.

ii. Argue that this amount of communication is essentially optimal (when, say, $N \approx (2^m)/10$ and $x = O(1)$).

Solution

The amount of communication needed is $xm + 1$ bits. Note that there are $\binom{N}{x}$ possible subsets that could have gone missing, so Sally needs at least $\log_2 \binom{N}{x}$ bits in order to resolve them (by the pigeonhole principle: otherwise there would be two possible choices for $S \setminus T$ for which Sally would receive the same message). When $N \gg x$, we have that $\log \binom{N}{x} \approx x \log(N)$. In more detail, we have:

$$\begin{aligned}
\log \binom{N}{x} &\geq \log \left(\frac{(N-x)^x}{x!} \right) \\
&\geq \log \left(\left(\frac{N-x}{x} \right)^x \right) \\
&= x \log \left(\frac{N-x}{x} \right) \\
&= x \log N - x \log \left(\frac{x}{1 - x/N} \right) \\
&= x \log N (1 - o(1)),
\end{aligned}$$

since the additive term $x \log(x/(1-x/N))$ is tiny compared to $x \log N$ when $x = O(1)$ and N is growing. Thus, the total amount of communication needed with *any* scheme is $x \log N(1 - o(1)) = xm(1 - o(1))$. So this is optimal even up to the leading constant. Cool!

4. **(Extra: RM Codes)** *[Probably we won't have time for this question...it's here in case you finish early and want some practice with RM codes too. This is not meant to be a tricky question, don't overthink it.]* Recall from the lecture videos/notes that $RM_2(m, r)$ is binary Reed-Muller code formed by m-variate polynomials of total degree at most r. For what m do there exist binary Reed-Muller codes of rate exactly $1/2$? For those m, what should r be?

Solution

These exist when m is odd, and $m = 2r + 1$. To see this, notice that the dimension of $RM_2(m, r)$ when $m = 2r + 1$ is

$$\sum_{j=0}^{r} \binom{m}{j} = 2^{m-1} = \frac{2^m}{2}.$$

5. **(Bonus: More RM Codes)** Come up with an algorithm for efficiently decoding RM codes up to half their distance. Here's an outline to get you started:

   (a) Suppose that $f(X_1, \ldots, X_m)$ is the polynomial corresponding to the codeword that was sent. Suppose that we get a corrupted codeword in $\mathbb{F}_2^{2^m}$. We can view this as some other (not necessarily low-degree) function $\tilde{f}(X_1, \ldots, X_m)$ from $\mathbb{F}_2^m$ to $\mathbb{F}_2$.

   (b) Let $\mathbf{X}$ denote $(X_1, \ldots, X_m)$, and for a set $S \subseteq [m]$, let $\mathbf{X}^S$ denote $\prod_{s \in S} X_s$.

(c) Explain why the following two things are true:

- For any $S \subseteq [m]$, $\sum_{\mathbf{a} \in \mathbb{F}_2^S} \mathbf{a}^S = 1$.
- For any $S \subseteq [m]$ and any $T \subsetneq S$, $\sum_{\mathbf{a} \in \mathbb{F}_2^S} \mathbf{a}^T = 0$.

(d) Fix some set S of size r (the degree of the RM code). Fix any $\mathbf{b} \in \mathbb{F}_2^{\bar{S}}$, where $\bar{S}$ denotes $[m] \setminus S$. For any $g : \mathbb{F}_2^m \to \mathbb{F}_2$, let $g_{\mathbf{b}} : \mathbb{F}_2^S \to \mathbb{F}_2$ be equal to $g(\mathbf{X})$, except we plug in $\mathbf{b}$ for the variables in $\bar{S}$.

Recall that $f(\mathbf{X})$ is our degree-r polynomial, and write

$$f(\mathbf{X}) = \sum_{S \subseteq [m] : |S| \leq r} c_S \mathbf{X}^S$$

for some coefficients $c_S \in \mathbb{F}_2$.

Explain why, for all $\mathbf{b} \in \mathbb{F}_2^{\bar{S}}$,

$$\sum_{\mathbf{a} \in \mathbb{F}_2^S} f_{\mathbf{b}}(\mathbf{a}) = c_S.$$

(e) Suppose that $\tilde{f}(\mathbf{X})$ corresponds to a corrupted codeword that differs from $f(\mathbf{X})$ on at most $2^{m-r-1} - 1$ values of $\mathbf{X}$ in $\mathbb{F}_2^m$. Explain why

$$\sum_{\mathbf{a} \in \mathbb{F}_2^S} \tilde{f}_{\mathbf{b}}(\mathbf{a}) = \sum_{\mathbf{a} \in \mathbb{F}_2^S} f_{\mathbf{b}}(\mathbf{a}) = c_S$$

for strictly more than half of the values of $\mathbf{b} \in \mathbb{F}_2^{\bar{S}}$.

(f) Use the above to design an algorithm...

Solution

This is describing *Reed's Decoder.*

(a),(b) Got it.

(c) For the first one, the only term that survives is $\mathbf{a} = \mathbf{1}$. For the second one, for any $i \in S \setminus T$, consider breaking the sum into pairs that are identical except on the i'th bit, and notice that these pairs cancel each other out.

(d) We can write

$$f_{\mathbf{b}}(\mathbf{X}) = c_S \mathbf{X}^S + \sum_{T \subsetneq S} \tilde{c}_T \mathbf{X}^T,$$

for some coefficients $\tilde{c}_T$. Then we apply the facts from the previous part to see that the first term is c_S, and the second is zero.

(a) Consider partitioning $\mathbb{F}_2^m$ into sets $I_{\mathbf{b}} = \{\mathbf{x} \in \mathbb{F}_2^m : \mathbf{x}|_{\bar{S}} = \mathbf{b}\}$. There are 2^{m-r} parts in this partition, and and most $2^{m-r}/2 - 1$ errors, which means that strictly less than half of these parts can have any errors in them. If there are no errors in $I_{\mathbf{b}}$, then

$$\sum_{\mathbf{a} \in \mathbb{F}_2^S} f_{\mathbf{b}}(\mathbf{a}) = \sum_{\mathbf{a} \in \mathbb{F}_2^S} \tilde{f}_{\mathbf{b}}(\mathbf{a}),$$

since the sum only involves evaluations of these functions in $I_{\mathbf{b}}$. The previous part implies these are both equal to c_S.

(b) We first recover c_S for all $|S| = r$, by taking the majority vote over all values of $\mathbf{b}$'s for each. (There are $2^{m-r} \leq n$ values of $\mathbf{b}$ for each S, and there are $\binom{m}{r}$ values of S). Then subtract

the corresponding terms off of $\tilde{f}$ and repeat for terms of degree $r-1$ and so on. At the end of the day, the running time is at most $n \cdot \sum_{j \leq r} \binom{m}{j} = nk = O(n^2)$.