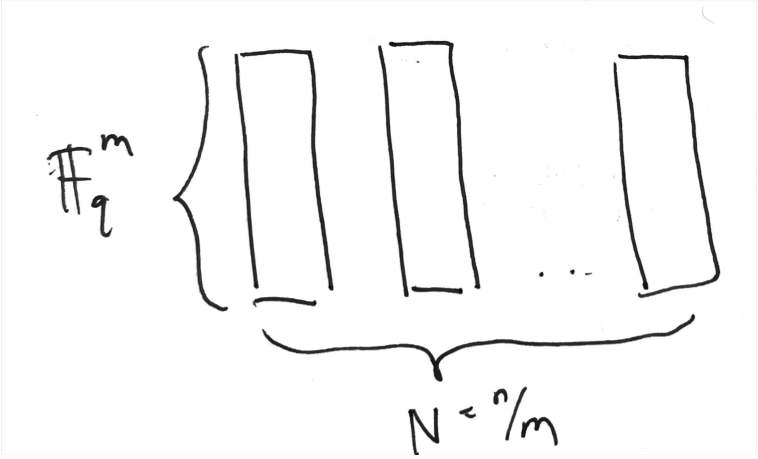
1 Recap - Folded Reed-Solomon codes

Last Class: Recall that last class we were list decoding the m-folded Reed Solomon (FRS) code, which took a Reed-Solomon codeword of length n over the alphabet $\mathbb{F}_q^m$ and repackaged it as a length N codeword over $\mathbb{F}_q^n$ as visualized here:



The condition we had on agreement t was:

$$\frac{t}{N} > \frac{Rm}{m-s+1} \cdot \frac{s}{s+1} + \frac{1}{s+1}$$

We then concluded that if you take $s \approx \frac{1}{\epsilon}, m \approx \frac{1}{\epsilon^2}$, then

$$\frac{t}{N} > R + \epsilon$$

allows us to achieve the condition, where R is the rate of the original Reed-Solomon code (our "repackaging" left the rate unchanged). The list size was q^{s-1}. Furthermore, we know we cannot do better than this, as this is the list-decoding capacity.

We can also view this code as a better way to correct bursty errors. If the errors occur in some burst smaller than m, then only one or two blocks get corrupted.

2 List Recovery

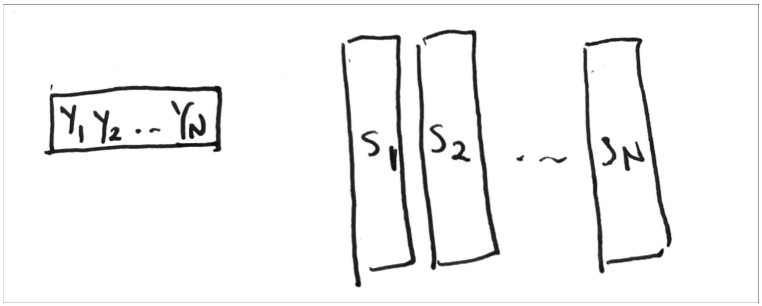
We now consider a related problem, a generalization of list decoding.

Definition 20.1. A code $C \subseteq \mathbb{F}_q^n$ is said to be (e, ℓ, L) list-recoverable if for any family of n subsets, $S_1, S_2, \ldots, S_n \subseteq \mathbb{F}_q$ with $\forall i, |S_i| \leq \ell$, we have:

$$\#\{c \in C : \#\{i : c_i \notin S_i\} \leq en\} \leq L$$

In other words,

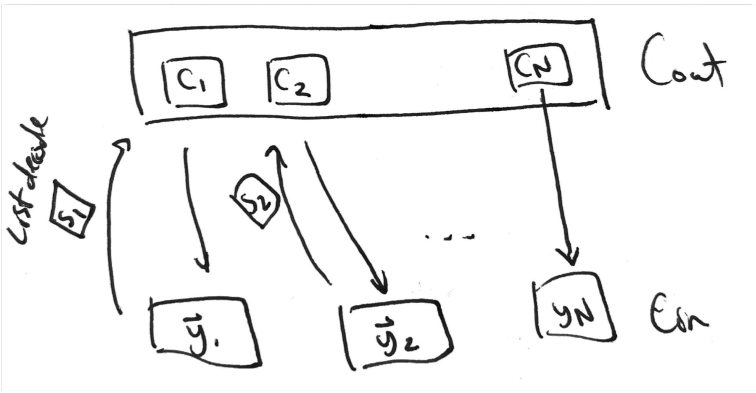
- For each place i, instead of a list of symbols y_i, we are given sets S_i of size at most ℓ. S_i contains the characters that we think that place could be.
- Our goal is to find all codewords $c \in C$ which incorporates one of these guesses at $\geq n(1-e)$ places. We hope this is a small list of size L.



Note that if $\ell = 1$, we return back to normal list decoding. Also, we naturally define the agreement as $t = n(1-e)$.

It turns out list recovery has numerous applications (see [NPR12] and [DW22] for a few examples).

Recall when we had (unique decoding) concatenated codes, we had an inner and outer code. Now suppose we list-decoded the inner codewords, then we would get sets S_i of possibilities for the c_i.



These constructions are quite useful.

Theorem 20.2. *Suppose we can list decode C_{in} up to radius ρ_{in} with list size ℓ and list recover C_{out} from ρ_{out} error fraction for ambiguity ℓ and output list of size L. Then the concatenated code can be list decoded up to a radius $\rho_{out}\rho_{in}$ with list size L.*

Proof. By the pigeonhole principle, if there are at most $\rho_{out}\rho_{in}$ errors, there are at most ρ_{out} blocks that have more than ρ_{in} places that are corrupted. Thus the number of incorrect inner codewords is at most ρ_{out}, which can successfully be list-recovered by C_{out}. □

3 FRS for List Recovery

For RS codes, $t > \sqrt{kn\ell}$ (there are suddenly ℓ times as many points to fit) so $e < 1 - \sqrt{R\ell}$. This means for the error fraction to even be positive, our rate must have $R < \frac{1}{\ell}$–our rate degrades as a function of the input list size.

It turns out that FRS works wonders for list recovery. For ambiguity ℓ, we lose a little bit, t needs to be a bigger.

Claim 20.3. *To make a Folded RS code list recoverable to ℓ places, we need:*

$$\frac{t}{N} > \frac{Rm}{m-s+1} \cdot \frac{s}{s+1} + \frac{\ell}{s+1}$$

Why is this the case? Recall the our original algorithm, which solved an interpolation problem. The only thing that changes is the number of interpolation conditions. We originally had $N(m-s+1)$ interpolation conditions, which were $Q(\gamma^{jm+p}, y_{jm+p}, \ldots, y_{jm+p+s-1}) = 0$ for $0 \leq p \leq m-s$ and $0 \leq j \leq n-1$. But now there are 1 of these for each element of S_j (different guesses for what this block could be), so there are at most $N(m-s+1)\ell$ interpolation conditions.

The rest of the previous argument works (setting the number of conditions to be less than the number of unknowns) we just get an extra factor of ℓ in the right term. If we pick $s \approx \ell/\epsilon, m \approx \ell/\epsilon^2$ then $t/N > R + \epsilon$ implies our condition (the proof is the exact same as the other case), so we can pick any arbitrary ℓ we want, without rate getting worse. Something has to give though, so notice the alphabet size gets bigger with bigger ℓ.

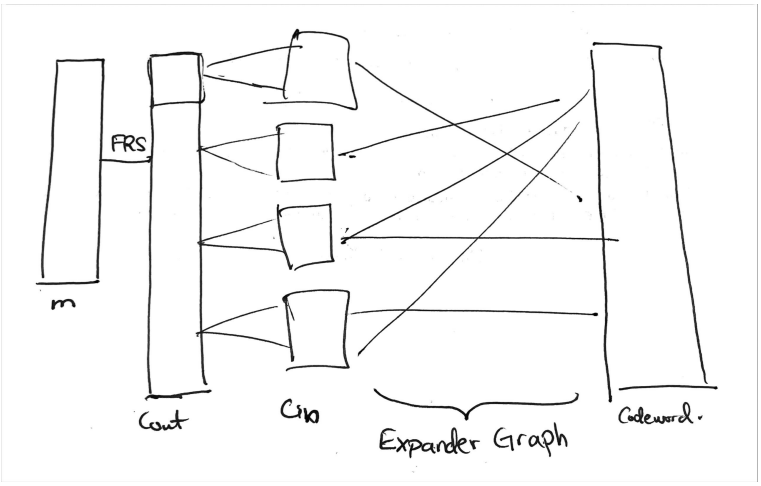
Imagine now concatenating FRS with an optimal brute-forced inner code. We get the following results.

1. We now have efficient binary concatenated codes that are list decodable up to the Zyablov bound [GR09]. That is,

   $$\forall \epsilon > 0, \rho(R) = \max_{R_{out}R_{in}=R} (1 - R_{out} - \epsilon)(h^{-1}(1 - R_{in}) - \epsilon)$$

   Recall that for unique decoding, we could only decode up to half this radius (the distance was equal to this).

2. The FRS alphabet size is huge! You need $|\Sigma| \geq n^{1/\epsilon^2}$ to get a radius $1 - R - \epsilon$. Using list recoverable FRS codes of rate $\to 1$, an inner code of optimal tradeoff over fixed algorithms, wherein $(1 - R - \epsilon, \text{poly}(1/\epsilon))$-list decodable code over $|\Sigma| = \exp(1/\epsilon)$ and an expander-based symbol redistribution,

   we get $(1 - R - \epsilon, n^{\text{poly}(1/\epsilon)}$-list decodable codes over alphabet size $\exp(\text{poly}(1/\epsilon))$ [GR08]. Note this is not much worse than the lower bound alphabet size of $\exp(1/\epsilon)$.

This is our ending point for list decoding, as we turn back to locally-decodable codes.

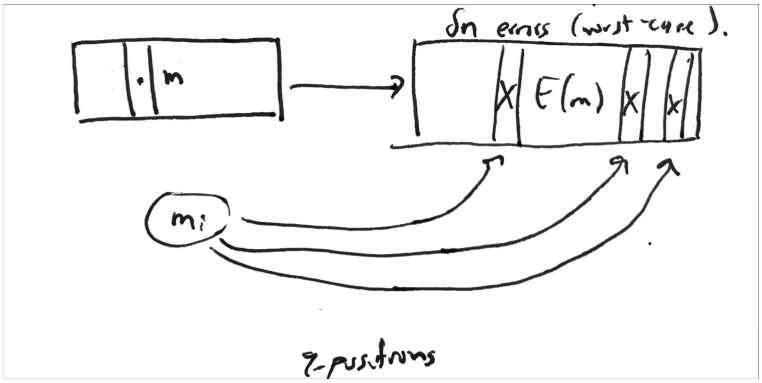
4 Limits on 2-query LDCs

Recall the definition of an LDC:

Definition 20.4. A (q, ϵ, δ)-LDC is a map $E : \{0,1\}^k \to \Sigma^n$ with a nonadaptive decoder Dec that makes q queries (i.e. the positions queried each time are the same) into $y \in \Sigma^n$ such that $\forall i \in [k], \forall m \in \Sigma^n, \forall y \in B(E(m), \delta n)$ we have:

$$Pr_{\text{decoder randomness}}[(\text{Dec}(y))_i = m_i] \geq \frac{1}{2} + \epsilon$$

The idea is that you only want one bit of the message, but you want to query q positions, which is a sublinear amount.



Note that you cannot get m_i with certainty if q is sublinear–the δn errors could've occured at those places. Furthermore, we need a randomized algorithm because of exactly this issue: an adversary can just corrupt all q places we check.

There is lots of work on different sublinear regimes. Here is a summary of some results.

Fact 20.5. *Consider a (q, ϵ, δ)-LDC over a message length k and codeword length n:*

- *If $q = O(1)$, then $n = \exp(k^{o(1)})$.*
- *If $q \approx \log^{1+\epsilon} n$, then $n = \text{poly}(k)$.*
- *If $q = n^{\epsilon}$, then we can actually approach rate 1.*

As we see, generally these codes do not have great rate, so they are pretty impractical for the real world. So instead, we rely on a related notion, called LRCs. These can stand for "Locally { Recoverable, Repairable, Reconstructible } Codes"; many disagree on what the acronym stands for.

Without loss of generality, we will now only discuss Linear LDCs, where the encoding map is a matrix. For example, consider the Hadamard 2-query LDC. The encoding map was $E : m \mapsto Had(m)$. The decoding algorithm picked a random r and did the following:

- $m_i \leftarrow m \cdot r + m \cdot (r + e_i)$
- $\hat{m}_i \leftarrow y_r + y_{r+e_1}$

We showed that

$$\mathop{\mathbf{Pr}}_{r}[\hat{m}_i = m_i] \geq 1 - 2\delta$$

The decoding/encoding is efficient and this only two queries! But recall it's rate is: $\frac{\log n}{n}$, which is pretty bad. But it's not Hadamard's fault.

Theorem 20.6 ([GKST02]). *Let $C : \mathbb{F}_2^k \to \mathbb{F}_2^n$ be a binary linear encoding that is a $(2, \epsilon, \delta)$-LDC. Then $n \geq 2^{\epsilon\delta k}$.*

The same result holds for general codes, too, but the proof is harder. [KdW02] gave the first argument for this, which is a quantum computing argument.

Before presenting the proof, we set up a lot of the groundwork. The first thing to do is define the notion of a smooth LDC.

4.1 Smooth LDCs

Suppose there are no errors and we encode m to some codeword $C(m)$. Then it is trivially easy to decode. Just use the C as the identity map and to read m_i, we read $C(m)_i$. To make the problem actually difficulty, let's instead say that you can't look at a message bit deterministically all of the time, but only with some probability.

Definition 20.7. An encoding map E is a (q, ϵ, c)-smooth LDC if it admits a decoding map Dec such that:

$$\forall i, \forall m, \mathbf{Pr}[Dec(C(m))_i = m_i] \geq \frac{1}{2} + \epsilon$$

and furthermore, for all j, decoder probes $C(m)_j$ with probability at most $\frac{c}{n}$.

It turns out a q-query LDC creates a q-query smooth LDC. In fact:

Lemma 20.8. *A (q, ϵ, δ)-LDC is also a $(q, \epsilon, \frac{q}{\delta})$-smooth LDC.*

Proof. We make an algorithmic reduction, over the same encoding map.

Smooth Dec(i):

- $S \leftarrow \{j : \mathbf{Pr}[\text{Dec}(i) \text{ queries } j] > \frac{q}{\delta n}\}$
- Call LDC algo Dec(i).
- Whenever Dec queries a position $j \in S$, tell it 0 instead of querying the position.
- Return the output of Dec.

Correctness: First off, note that any location cannot be queried with probability more than $\frac{q}{\delta n}$, because if it were going to be, the smooth decoder would not actually allow the query to go through. Furthermore, there are only at most $\frac{\delta n}{q} < \delta n$ "errors" fed to the LDC decoder, so we inherit the same correctness probability. □

Furthermore, the following fact is true.

Exercise 20.9. A (q, ϵ, c)-smooth LDC is a $(q, \epsilon - qc\delta, \delta)$ LDC.

4.2 Simplifications for the 2-query case

Suppose our (linear smooth LDC) decoding algorithm picked positions j_1 and j_2. We can only get $m_i = m \cdot e_i$ from $m \cdot a_{j_1}$ and $m \cdot a_{j_2}$ only when $e_i \in \text{span}\{a_{j_1}, a_{j_2}\}$ (otherwise no linear combination of the information can get us the answer). This might mean a_{j_1} or a_{j_2} is e_i.

Claim 20.10. *We can ignore this case.*

Proof. Let $p_i = \{j \mid a_j = e_i\}$. The number of i such that $|p_i| > \frac{2n}{k}$ is at most $\frac{k}{2}$ by the pigeonhole principle. Let's puncture the message space to remove $k/2$ indices, including those indices where p_i is as large as above. This will give you a code from $\mathbb{F}_2^{k/2} \to \mathbb{F}_2^n$ which such that for all i, $|p_i| \leq \frac{2n}{k}$. Then $\mathbf{Pr}[\text{query inside } p_i] \leq \frac{2}{\delta n} \cdot \frac{2n}{k} = \frac{4}{\delta k} \ll \frac{\epsilon}{2}$ if k is large: so we have at least an $\epsilon/2$ advantage. □

Next time, we will tackle the other case: $a_{j_1} + a_{j_2} = e_i$.

References

[DW22] Dean Doron and Mary Wootters. High-Probability List-Recovery, and Applications to Heavy Hitters. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 2

[GKST02] O. Goldreich, H. Karloff, L.J. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 175–183, 2002. 20.6

[GR08] Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008. 2

[GR09] Venkatesan Guruswami and Atri Rudra. Better binary list decodable codes via multilevel concatenation. *IEEE Transactions on Information Theory*, 55(1):19–26, 2009. 1

[KdW02] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument, 2002. 4

[NPR12] Hung Ngo, Ely Porat, and Atri Rudra. Efficiently decodable compressed sensing by list-recoverable codes and recursion. volume 14, pages 230–241, 02 2012. 2