1 LDC Constructions

1.1 Introduction

In the first part of lecture, we will see Gopalan's presentation of a particular LDC construction [Gop09], that achieves 3-query LDCs with subexponential block size in the dimension, in contrast to 2-LDCs which require exponential block size. For ease of exposition, in this lecture we will describe how to construct a 6-query LDC rather than a 3-query LDC. This construction is based on the following combinatorial object:

Definition 22.1 (Matching Vector Family)**.** A collection of vectors $\underline{u_1}, \ldots, \underline{u_M}$ and $\underline{v_1} \ldots, \underline{v_M}$ in $\mathbb{Z}_m^n$ is a Matching Vector Family if $\underline{u_i} \cdot \underline{v_i} = 0$ for all $i \in [M]$ and $\underline{u_i} \cdot \underline{v_j} \neq 0$ for all $i, j \in [M]$ with $i \neq j$.

Theorem 22.2 ([Gro00])**.** *For $m = 6$, there is an explicit Matching Vector Family with $M \geq 2^{\tilde{\Omega}((\log n)^2)}$.*

1.2 Construction

Before defining the code, we introduce some parameters and notation. Let $\mathbb{F}_q$ be a finite field, and set $m = q - 1$. Let g be an arbitrary generator of the multiplicative group $\mathbb{F}_q^*$, so $\mathbb{F}_q^* = \{1, g, g^2, \ldots, g^{m-1}\}$. Notice that the generator fixes an isomorphism from $\mathbb{Z}_m$ to $\mathbb{F}_q^*$ given by $i \to g^i$. For a 6-query LDC, it will suffice to consider the case $q = 7$ and $m = 6$, and we will only consider these values for this lecture.

For $x \in \mathbb{F}_q$ and an integer vector $\underline{w} \in \mathbb{Z}^n$, define $x^{\underline{w}} := (x^{w_1}, x^{w_2}, \ldots, x^{w_n})$. For vectors $\underline{x} \in (\mathbb{F}_q)^n$ and $\underline{w} \in \mathbb{Z}^n$, define $\underline{x}^{\underline{w}} := x_1^{w_1} x_2^{w_2} \ldots x_n^{w_n}$. Fix a Matching Vector Family $\{\underline{u_i}\}_{i \in [M]}$ and $\{\underline{v_i}\}_{i \in [M]}$ with the guarantees of Theorem 22.2. We further define $\chi_i(\underline{x}) := \underline{x}^{\underline{v_i}}$ for $i \in [M]$.

The code's message space will be $\mathbb{F}_q^M$, and the code space will be $\mathbb{F}_q^{(\mathbb{F}_q^*)^n}$. For a message $\lambda \in \mathbb{F}_q^M$, define the n-variate polynomial

$$P_\lambda(x_1, \ldots, x_n) = \sum_{i \in [M]} \lambda_i \underline{x}^{\underline{v_j}} = \sum_{i \in [M]} \lambda_i \chi_i(x) \, .$$

Encode λ as the list of evaluations of P_λ on points in $(\mathbb{F}_q^*)^n$, i.e. $\mathrm{Enc}(\lambda) = P_\lambda(a)|_{a \in (\mathbb{F}_q^*)^n}$.

1.3 Local Decoding of the Code

In order to prove local decodability, we will demonstrate a perfectly smooth local decoder that can recover λ_i from m smooth queries to $\mathrm{Enc}(\lambda)$. This entails designing m randomized queries, with each query individually being uniformly random, that can recover λ_i. We'll first show how to do this with m deterministic queries.

The main idea is that we can access the i^{th} coefficient of P_λ by evaluations of P_λ at vectors of the form $g^{\underline{u_i}}$. To build up to this idea, we make the following simple observation.

Observation 22.3. For any $\underline{w} \in \mathbb{Z}_m^n$ and $j \in [M]$, $\chi_j(g^{\underline{w}}) = g^{\underline{w} \cdot \underline{v_j}}$. In particular, $\chi_j(g^{\underline{u_i}})$ equals 1 if and only if $i = j$.

In order to access λ_i, we would like to isolate the i^{th} term by zeroing out all other terms where $j \neq i$. There is a standard trick we can use to do this – namely, if $b \in \mathbb{F}_q^*$ with $b \neq 1$, then $1 + b + \ldots + b^{m-1} = 0$. Applying this to $b = \chi_j(g^{\underline{w}})$, we get

$$\sum_{\alpha \in \mathbb{Z}_m} \chi_j(g^{\alpha \underline{u_i}}) = \sum_{\alpha \in \mathbb{Z}_m} \chi_j(g^{\underline{u_i}})^\alpha = \begin{cases} -1 \text{ if } i = j, \\ 0 \text{ if } i \neq j. \end{cases} \tag{22.1}$$

Therefore,

$$\sum_{\alpha \in \mathbb{Z}_m} P_\lambda(g^{\alpha \underline{u_i}}) = \sum_{j \in [M]} \lambda_j \sum_{\alpha \in \mathbb{Z}_m} \chi_j(g^{\alpha \underline{u_i}}) = \sum_{j \in [M]} \lambda_j \cdot (-\delta_{ij}) = -\lambda_i \, . \tag{22.2}$$

This tells us that we can recover λ_i using m queries at the points $\{g^{\alpha \underline{u_i}}\}_{\alpha \in \mathbb{Z}_m}$. However, this is not yet enough to prove local decodability because the queries aren't smooth – in fact they're completely deterministic.

We can convert each query above into a uniformly distributed query over $(\mathbb{F}_q^*)^n$ by leveraging the fact that $\chi_j(x \cdot y) = \chi_j(x) \cdot \chi_j(y)$. Concretely, we can query at a random "line" parallel to the original queries, which consists of the points $\{g^{\underline{y} + \alpha \underline{u_i}}\}_{\alpha \in \mathbb{Z}_m}$ for a random $\underline{y} \sim \mathbb{Z}_m^n$. For these queries, Eqn. 22.1 turns into

$$\sum_{\alpha \in \mathbb{Z}_m} \chi_j(g^{\underline{y} + \alpha \underline{u_i}}) = \chi_j(g^{\underline{y}}) \sum_{\alpha \in \mathbb{Z}_m} \chi_j(g^{\underline{u_i}})^\alpha = \begin{cases} -\chi_j(g^{\underline{y}}) \text{ if } i = j, \\ 0 \text{ if } i \neq j. \end{cases}$$

And Eqn. 22.2 turns into

$$\sum_{\alpha \in \mathbb{Z}_m} P_\lambda(g^{\underline{y} + \alpha \underline{u_i}}) = \sum_{j \in [M]} \lambda_j \sum_{\alpha \in \mathbb{Z}_m} \chi_j(g^{\underline{y} + \alpha \underline{u_i}}) = \sum_{j \in [M]} \lambda_j \cdot (-\delta_{ij} \chi_j(g^{\underline{y}})) = -\lambda_i \chi_i(g^{\underline{y}}) \, .$$

Since we know $\underline{y}$, we can sum the output of queries at the m points $\{g^{\underline{y} + \alpha \underline{u_i}}\}_{\alpha \in \mathbb{Z}_m}$ and divide that by $-\chi_i(g^{\underline{y}})$ to obtain λ_i. For $\underline{y} \sim \mathbb{Z}_m^n$, $g^{\underline{y}}$ is a uniformly random element of $(\mathbb{F}_q^*)^n$, so the α^{th} query $g^{\underline{y} + \alpha \underline{u_i}} = g^{\underline{y}} g^{\alpha \underline{u_i}}$ is distributed uniformly among the codeword bits. This concludes the proof that the code has a perfectly smooth m-query local decoder. By a union bound argument, this also implies that the above code is a $(m, \delta, 1 - m\delta)$-LDC, since if δ fraction of codeword bits are corrupted, then the m queries avoid all corrupted codewords except with probability at most $m\delta$.

1.4 Parameters and Other Notes

The code's dimension is M, which is $2^{\tilde{\Omega}((\log n)^2)}$ by Theorem 22.2. This implies $n = 2^{\tilde{O}(\sqrt{\log M})}$. On the other hand, the block length is $(q-1)^n = 2^{O(n)} = 2^{2^{\tilde{O}(\sqrt{\log M})}} = 2^{M^{o(1)}}$, which is subexponential in the dimension. In contrast, replacing Theorem 22.2 by a "trivial" construction of Matching Vector Families would give block length of $2^{M^{1/5}}$ which is much larger.

A version of the above argument can be made to work for any m that divides $q - 1$ (not just $m = q - 1$), so any version of Theorem 22.2 for smaller m would give m-query LDC constructions, and indeed this can be used to construct 4-query LDCs with similar rate. Getting 3-query LDCs requires some more ideas.

2 Matching Vector Families

2.1 OR Representations

We will briefly elaborate on how large Matching Vector Families like those in Theorem 22.2 are constructed from algebraic objects known as OR Representations.

Definition 22.4 (OR Representation)**.** An integer polynomial $p(x_1, \ldots, x_\ell)$ represents the OR function modulo m if:

1. $p(0^\ell) = 0 \mod m$
2. For all $x \in \{0,1\}^\ell$ with $x \neq 0^\ell$, $p(x) \neq 0 \mod m$.

Here we think of m as a constant and ℓ as increasing. As an example, the polynomial $p(x) = 1 - \prod_{i \in [\ell]}(1 - x_i)$ is exactly the OR function on $\{0,1\}^\ell$, and is therefore an OR representation modulo any $m \geq 2$. However, the degree of p is ℓ, which is too high for applications.

Notice that this polynomial exactly equals 1 on nonzero inputs in $\{0,1\}^\ell$, but the definition allows us the flexibility to have it equal any nonzero value. It turns out that for certain m, we can utilize this flexibility to construct OR representations of much lower degree.

Theorem 22.5 ([BBR94])**.** *There exist explicit OR representations modulo m of degree $O(\ell^{1/r})$, where r is the number of distinct prime factors of m.*

For our case of $m = 6$, this gives us degree $O(\sqrt{\ell})$ OR representations modulo m. As a side note, the theorem is tight for $r = 1$, as we need $\Omega(\ell)$ degree whenever m is a prime power, which is the reason we can't immediately get 4-query LDCs using a similar construction.

Claim 22.6. *If m is a prime power, any OR representation modulo m has degree at least $\ell/(m-1)$.*

Proof. Let p be an OR representation modulo m. By Fermat's little theorem, p^{m-1} exactly computes the OR function modulo m, so it must have degree at least ℓ. Therefore, p has degree at least $\ell/(m-1)$. □

2.2 Matching Vector Families from OR Representations

We describe how to obtain large Matching Vector Families from low degree OR representations. Set $m = 6$. Let Q be an OR representation of degree $d = O(\sqrt{\ell})$ as promised by Theorem 22.5. Without loss of generality, Q is multilinear as we are only concerned with its values on boolean inputs. For $y \in \{0,1\}^\ell$, define $Q_y(x) = Q(x \oplus y) \mod m$.

We will construct a pair of matching vectors for each $y \in \{0,1\}^\ell$. Each vector will be indexed by multilinear monomials in ℓ variables of degree $\leq d$.

Set $\underline{v_y}$ to be the coefficient vector of Q_y, and set $\underline{u_y}$ to be the vector of evaluations of all degree $\leq d$ multilinear monomials at y. We have $\underline{v_y} \cdot \underline{u_z} = Q_y(z)$ by definition, so $\underline{v_y} \cdot \underline{u_y} = Q_y(y) = 0$ and $\underline{v_y} \cdot \underline{u_z} = Q_y(z) \neq 0$ for all $y \neq z$ by definition of OR representations. So this is a valid Matching Vector Family.

The number of multilinear monomials of degree $\leq d$ is $\sum_{j=0}^{d} \binom{\ell}{j} = \ell^{O(\sqrt{\ell})}$. Therefore $n \leq \ell^{O(\sqrt{\ell})}$, that is, $\ell \geq \tilde{\Omega}((\log n)^2)$. On the other hand, $M = 2^\ell$, which is at least $2^{\tilde{\Omega}((\log n)^2)}$, giving us the conclusion of Theorem 22.2.

3 Locally Repairable Codes

For the rest of the lecture, we will pivot to a new topic. The inspiration for LRCs/Locally Repairable Codes comes from the setting of distributed storage systems. These systems involve a collection of servers that cumulatively store some data. The analogy to coding theory is that the data corresponds to the message, and the state of a server corresponds to a codeword bit. In practice, distributed storage systems need to be tolerant to two kinds of errors:

1. The relatively common case of having a small number of servers crash, i.e. a small number of erasures. For simplicity, we will think of this case as just one erasure.
2. The rare case of catastrophic failure, i.e. a large number of erasures.

We would like to be able to handle the common case of one erasure much more efficiently (in terms of network bandwidth) than the rare case. We will enforce efficiency by requiring the recovery algorithm for one erasure to be very local.

Definition 22.7. A $[n, k]$ code $C \subseteq \Sigma^n$ is an (r, d)-LRC if

1. C can recover from a single erasure by querying at most r bits. That is, for all $i \in [n]$, there is a set $R_i \subseteq [n] - i$ with $|R_i| \leq r$ such that c_i can be recovered from $c_{|R_i}$.
2. $\text{dist}(C) \geq d$.

For linear codes, we remark that the first requirement can be written in terms of parity checks.

Observation 22.8. If C is linear, then condition 1 in Definition 22.7 is equivalent to "there is a parity check satisfied by C with support $\{i\} \cup R_i$."

The central question with LRCs is to obtain optimal tradeoffs between the parameters n, d, r, and k. In this lecture, we will see an optimal construction of a weaker object known as Message LRCs, which we will abbreviate as MLRCs.

Definition 22.9. Let C be a $[n, k]$ code with a systematic encoding map $\text{Enc} : \Sigma^k \to \Sigma^n$, so $\text{Enc}(x)_{\leq k} = x$. C is a (r, d)-MLRC if

1. For all $i \in [k]$, there is a set $R_i \subseteq [n] - i$ with $|R_i| \leq r$ such that c_i can be recovered from $c_{|R_i}$.
2. $\text{dist}(C) \geq d$.

MLRCs are related to LRCs in the same way that LDCs are related to LCCs. It is quite straightforward to construct a MLRC that is locally resistant to one erasure, for example, one can take any distance d code and append the message to it. It is also straightforward to construct optimal MLRCs over large alphabets.

Lemma 22.10. *There is an explicit (r, d)-MLRC with $d = n - k - \lceil k/r \rceil + 2$.*

Proof. We start from a $[n', k, n' - k + 1]_q$ Reed-Solomon code C' for $n' = n - \lceil k/r \rceil + 1$. This code already has distance $n' - k + 1 = n - k - \lceil k/r \rceil + 2$ as desired. We modify this code to also satisfy requirement 1 in Definition 22.9.

Consider the generator matrix for a systematic encoding of C':

$$\begin{bmatrix} I_k \\ a_1 \\ \vdots \\ a_{n-k} \end{bmatrix}$$

where I_k is the $k \times k$ identity matrix, and $a_1, \ldots, a_{n-k} \in \mathbb{F}_q^k$ are the rest of the rows. For the Reed-Solomon code, we can obtain $a_1, \ldots, a_{n-k}$ that have full support. In order to locally correct from an erasure of one of the e_i rows within the I_k section, we want to find a subset of at most r rows $R_i \subseteq [k] - i$ such that e_i is in the span of R_i.

To achieve this, we will replace the row a_{n-k} with $\lceil k/r \rceil$ rows with support $\leq r$ each. Partition the indices $[k]$ into sets $I_1, \ldots, I_{\lceil k/r \rceil}$ each of size at most r. Write $a_{n-k} = \sum_{i \in \lceil k/r \rceil} b_j$, where each b_j has support equal to I_j (we can do this because a_{n-k} has full support). Let C be the code obtained by replacing a_{n-k} with the b_js, so it has the following generator matrix:

$$\begin{bmatrix} I_k \\ a_1 \\ \vdots \\ a_{n-k-1} \\ b_1 \\ \vdots \\ b_{\lceil k/r \rceil} \end{bmatrix}$$

We must argue that C satisfies the requirements for a (r, d)-MLRC.

1. Say we want to recover from an erasure at position i, where $i \in I_j$ for some j. To do this, we set $R_i = \{b_j\} \cup \{e_{i'} : i' \in I_j - i\}$, and notice $b_j - \sum_{i' \in I_j} b_j[i'] = b_j[i] \cdot e_i$, so we can so we can recover the i^{th} message bit from the codeword bits in R_i. As required, we have $|R_i| = 1 + (r-1) = r$.
2. The distance of C is at least the distance of C', because if a message m satisfies $\langle m, b_j \rangle = 0$ for all j, then $\langle m, a_{n-k} \rangle = 0$. Therefore $\text{dist}(C) \geq \text{dist}(C') = d$.

□

We make two remarks:

- This construction is tight, and it achieves the analogue of the Singleton bound one can prove for MLRCs which takes the form $d \leq n - k - \lceil k/r \rceil + 2$.
- This construction was a very simple hacky modification to Reed-Solomon codes. This is not the case for LRCs – constructions of good LRCs are much more subtle.

References

[BBR94] David A Barrington, Richard Beigel, and Steven Rudich. Representing boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4(4):367–382, 1994. 22.5

[Gop09] Parikshit Gopalan. A note on efremenko's locally decodable codes. In *Electron. Colloquium Comput. Complex.*, volume 16, page 69, 2009. 1.1

[Gro00] Vince Grolmusz. Superpolynomial size set-systems with restricted intersections mod 6 and explicit ramsey graphs. *Combinatorica*, 20(1):71–86, 2000. 22.2