18.408 Topics in Theoretical Computer Science Fall 2022 Lectures 2 and 3

Dor Minzer

The topic for today is error correcting codes, which can be thought of as combinatorial analogs of PCPs. We will give some basic definitions and examples of error correcting codes, as well as discuss notions such as local testability and concatenation/ composition of error correcting codes. In particular, we will show how to construct an explicit error correcting code with constant rate, relative distance and binary alphabet. This process is analogous to the process we will take in future lectures in the proof of the PCP theorem (but simpler). In particular, PCP considerations will motivate us to discuss the notions of locally testable codes, locally decodeable codes and random self-correction.

1 Error Correcting Codes

Let Σ be a finite alphabet, and let $n \in \mathbb{N}$ be thought of as very large. An error correcting code is a collection of vectors $C \subseteq \Sigma^n$. The rate of an error correcting code measures how dense C is in Σ^n in logarithmic scale, i.e. $R = \frac{\log(|C|)}{\log(|\Sigma^n|)}$, and the distance of the code C is defined to be

$$d(C) = \min_{x,y \in C, x \neq y} \Delta(x,y), \qquad \text{where} \quad \Delta(x,y) = \#\left\{ i \in [n] \mid x_i \neq y_i \right\}.$$

Thus, the relative distance of C is defined as $d_{\mathsf{relative}}(C) = \frac{d(C)}{n}$. Often times, the length of a codeword, n, is referred to as the block length. With these parameters in mind, we often refer to C as an $(n, d_{\mathsf{relative}}(C), R(C), q)$-code.

Intuitively, it is clear that there is tension between the relative distance of a code and the rate of the code. The denser the code is, the less "space" there can be between distinct codewords. Indeed, many results in coding theory are concerned with exactly nailing down the tradeoff between parameters error correcting codes can achieve. For example, one of the most basic results of this form is the *Hamming bound.* Define $B_x = \left\{ z \in \Sigma^n \mid \Delta(x,z) \leqslant \frac{1}{2}d(C)n - 1 \right\}$ for each $x \in C$, namely the ball around x of radius $\frac{1}{2}d(C)n - 1$. Note that by the definition of relative distance, the balls B_x and B_y are disjoint for any two distinct points $x, y \in C$, and so

$$|\Sigma^n| \geqslant |\cup_{x \in C} B_x| = \sum_{x \in C} |B_x| \geqslant |C| \binom{n}{\frac{1}{2}d(C)n - 1} |\Sigma|^{\frac{1}{2}d(C)n - 1}.$$

Simplifying, one sees that $R + \frac{d(C)}{2} + o(1) \leqslant 1$. Thus, it is already clear that a code cannot have both relative distance and rate being close to 1.

Our interest in error correcting codes in this course is somewhat different, and we will not care so much about obtaining tight tradeoffs between parameters. For us, it will often be good enough that a code has constant relative distance, and not-too-terrible rate.

One particularly important class of codes is the class of linear error correcting codes. In this context, the alphabet Σ is equipped with an algebraic structure, and more precisely a field, say $\Sigma = \mathbb{F}_q$, where q is a power of a prime number.

Definition 1.1. *A linear error correcting code is a subset $C \subseteq \mathbb{F}_q^n$ which is a subspace of $\mathbb{F}_q^n$ over $\mathbb{F}_q$.*

The distance of a linear error correcting code. Let $C \subseteq \mathbb{F}_q^n$ be a linear error correcting code. Note that since C is a subspace we have $0 \in C$, so the distance of the code always satisfies

$$d(C) = \min_{x \neq y \in C} \Delta(x,y) \geqslant \min_{x \in C, x \neq 0} \Delta(x,0) = \min_{x \in C \setminus \{0\}} |\mathsf{supp}(x)|\,.$$

In fact, this can be observed to be an equality. For any x, y we have that $\Delta(x,y) = \Delta(x-y,0)$, and as C is a subspace, if $x, y \in C$ are distinct then $x - y \in C \setminus \{0\}$. Thus we get that:

Claim 1.2. *For a linear error correcting code $C \subseteq \mathbb{F}_q^n$ we have that $d(C) = \min_{x \in C \setminus \{0\}} |\mathsf{supp}(x)|$.*

In words, the distance of a linear error correcting code is the minimum Hamming weight of a non-zero codeword from it.

The rate as dimension. The rate of a linear error correcting code C also takes a special meaning. Let $k = \mathsf{dim}(C)$; observe that $|C| = q^k$, so $R(C) = \frac{\log|C|}{\log q^n} = \frac{k}{n}$. Hence the rate of C is the dimension of C as a subspace divided by the dimension of the ambient vector space. We will often times denote $k = \mathsf{dim}(C)$, and refer to k as the dimension of the code.

Generating matrices. Since any linear correcting code C is a subspace, we can view it as the image of some linear transformation; more precisely, there is a matrix $M \colon \mathbb{F}_q^k \to \mathbb{F}_q^n$ so that $\mathsf{Image}(M) = C$. Indeed, one way to construct M is by picking a basis $v_1, \ldots, v_k$ of C and defining $Me_i = v_i$. Such matrix M is called a *generating matrix* of C. Written otherwise, we say $M \in \mathbb{F}_q^{n \times k}$ is a generating matrix of C if

$$C = \left\{ Mz \mid z \in \mathbb{F}_q^k \right\}.$$

Generating matrices are quite useful, as they succinctly represent a code (which consists of many codewords). Also, one can view them as "encoding algorithms"; namely, if we have some message $z \in \mathbb{F}_q^k$ that we wish to send over a noisy channel and we want to make it more robust against the noise, we can compute $x = Mz$ and send it instead.

One of the primary goals of coding theory is to construct codes with "as good parameters as possible", and we have already seen that it is impossible to have both the relative distance and rate to be simultaneously close to 1. Is it at least possible to get them both to be bounded away from 0, though? It turns out that if one is willing to compromise on the alphabet size to be large, it is possible (and in fact not hard).

1.1 The Reed-Solomon Code

Let $d \in \mathbb{N}$ be a parameter, take $q \geqslant n$, and take distinct points $a_1, \ldots, a_n \in \mathbb{F}_q$. The Reed-Solomon code of degree d over points $a_1, \ldots, a_n$ is defined as

$$\mathsf{RS}_{d,a_1,\ldots,a_n,q} = \{ (f(a_1), \ldots, f(a_n)) \mid f \colon \mathbb{F}_q \to \mathbb{F}_q \text{ is a polynomial of degree at most } d \}\,.$$

In other words, for each function $f\colon \mathbb{F}_q \to \mathbb{F}_q$ of the form $f(x) = \sum_{i=0}^{d} \alpha_i x^i$ we have a word in $\mathsf{RS}_{d,a_1,\ldots,a_n,q}$, which is the vector of evaluations of it on the points $a_1,\ldots,a_n$. Often times, we will have $q = n$ in which case $\mathbb{F}_q = \{a_1,\ldots,a_n\}$.

Next, we calculate the parameters of $\mathsf{RS}_{d,a_1,\ldots,a_n,q}$. Clearly, the block length of the code is n, and the alphabet size is q. As for the rate, we have that the number of polynomials of degree at most d is q^{d+1} since we have q options to choose each one of the coefficients, hence the rate of the code is $R(\mathsf{RS}_{d,a_1,\ldots,a_n,q}) = \frac{\log\left(q^{d+1}\right)}{\log(q^n)} = \frac{d+1}{n}$. Therefore, if we want the rate to be constant, we need to take d to be linear in n.

Finally, for the distance of the code, by Claim 1.2 it suffices to bound the number of roots a non identically 0, univariate polynomial of degree at most d has. The fundamental theorem of algebra tells us that this is at most d, so $d(\mathsf{RS}_{d,a_1,\ldots,a_n,q}) = n - d$ and so $d_{\mathsf{relative}}(\mathsf{RS}_{d,a_1,\ldots,a_n,q}) = \frac{n-d}{n} = 1 - \frac{d}{n}$. Hence, the code $\mathsf{RS}_{d,a_1,\ldots,a_n,q}$ is an $(n, 1 - \frac{d}{n}, \frac{d+1}{n}, q)$ code, for $q \geqslant n$.

We see that taking $d = n/2$ for example, we get that both the rate and the relative distance are constant, which is great! However, as we are forced to have at least n points in our field, the alphabet size we get is quite large. Can we shrink the alphabet size somehow, and (ideally) construct an error correcting code over bits $\{0,1\}$ with constant relative distance and rate?

Later on in the course we will see an analogous situation, wherein without too much work we will able to construct PCPs for which the gap between the completeness and soundness is large (analogously to the distance of the code above), but the alphabet size grows as a result of this operation. We will want to shrink the alpahbet size. It is worth noting that in PCPs, the rate parameter is analogous to the size of the PCPs, and there it will be less important for us that it is constant (this would correspond to linear size PCPs, and we will only aim at polynomial size PCPs).

1.2 Composition of Codes

To answer this question, in this section we present the technique of code concatenation/ composition. In the coding theory literature, this operation is known as concatenation, but we think that the word composition makes more sense and will henceforth use this terminology.

Suppose we have two codes C_1, C_2 which are (n_1, d_1, r_1, q_1) and (n_2, d_2, r_2, q_2) codes. Further suppose that the number of codewords in C_2 is at least the alphabet size of C_1, i.e. $|C_2| \geqslant q_1$. In this situation, we can construct a composed code $C_1 \circ C_2$, as follows. Fix some injective map $M\colon \mathbb{F}_{q_1} \to C_2$; that is, for each alphabet symbol σ of C_1 choose some distinct codeword $c_2 \in C_2$, and map $M\sigma = c_2$. The idea in the composed code $C_1 \circ C_2$ then is to take each codeword in C_1 and replace each symbol in it with its corresponding codeword in C_2. Namely, the composed code is

$$C_1 \circ C_2 = \{\, (M(x_1),\ldots,M(x_{n_1})) \mid (x_1,\ldots,x_n) \in C_1 \} \,.$$

Next, we calculate the parameters of this code. Note that the block length is $n_1 \cdot n_2$, the alphabet is $\mathbb{F}_{q_2}$, and number of codewords in $C_1 \circ C_2$ is the same as the number of codewords in C_1 so

$$R(C_1 \circ C_2) = \frac{\log(|C_1|)}{\log(q_2^{n_1 n_2})} = \frac{\log(q_1^{n_1}) R(C_1)}{\log(q_2^{n_1 n_2})} = R(C_1)\frac{\log(q_1)}{\log(q_2^{n_2})} = R(C_1)R(C_2),$$

where the last transition holds if $q_1 = |C_2|$.

As for the distance, if we take distinct $(x_1,\ldots,x_n) \in C_1$ and $(y_1,\ldots,y_n) \in C_1$, then there are at least $d(C_1)$ indices i such that $x_i \neq y_i$, and then $M(x_i)$ and $M(y_i)$ will be different in at least $d(C_2)$ locations. Thus, $d(C) \geqslant d(C_1)d(C_2)$, and so $d_{\mathsf{relative}}(C) \geqslant d_{\mathsf{relative}}(C_1)d_{\mathsf{relative}}(C_2)$.

Summarizing, the code $C_1 \circ C_2$ is an $(n_1 n_2, d_1 d_2, r_1 r_2, q_2)$ code. This means that if the relative distances and rates of both C_1 and C_2 are constant, then the same holds for the composed code (albeit with a somewhat worse constant); furthermore the alphabet of $C_1 \circ C_2$ is inherited from the code C_2, which is potentially much smaller than that of C_1.

We also remark that if the map M is itself a linear map, then $C_1 \circ C_2$ is also a linear code, hence the composition operation works well with respect to linearity of codes. Composition also enjoys other properties that we will explore later on in the course.

Example. While simple, this operation is very powerful, and we will next see how to use the Reed-Solomon code with itself to get codes with constant rate and distance and smaller alphabet size. Indeed, take $C_1 = \mathsf{RS}_{d,a_1,\ldots,a_n,q}$ for $q = n$ and $d = n/2$ which is a $(n, 1/2, 1/2, n)$ code, and $C_2 = \mathsf{RS}_{d',a'_1,\ldots,a'_{n'},q'}$ for $n' = q' = \log n$, $d' = q'/2$ which is a $(\log n, 1/2, 1/2, \log n)$ code that has at least $(q')^{d'} > n$ codewords. Hence, we may compose these codes and get an $(n \log n, \frac{1}{4}, \frac{1}{4}, \log n)$ code. We can in fact repeat this idea a few more items; for example, doing it once more yields an $(n \log n \log\log n, \frac{1}{8}, \frac{1}{8}, \log\log n)$ code, so we can shrink the alphabet to be very small.

This idea alone would never really bring us down to a constant size alphabet. However, once the alphabet size is small enough ($q = \log\log n$ will do), one can just find a binary code with constant rate and relative distance by brute force. Indeed, if we take $n' = K \log q$ for a large absolute constant K, we can construct a code $C \subseteq \mathbb{F}_2^{n'}$ by going over all vectors $x \in \mathbb{F}_2^{n'}$ and at each time, adding x to C if doing so would not decrease the relative distance of C below 0.01. The running time of this is $2^{O(n')} < \mathsf{poly}(n)$, and clearly when the process terminates we get a code C with relative distance at least 0.01. Next, we argue that C also has a constant rate. Indeed, when the process ends, for each $x \in \mathbb{F}_2^{n'}$, the ball $B_x = \left\{ z \in \mathbb{F}_2^{n'} \,\middle|\, \Delta(x,z) \leqslant 0.01 n' \right\}$ contains some point from C (otherwise we could add x to C). Thus, letting B be any one of these balls, we have

$$|C| \geqslant \frac{2^{n'}}{|B|} \geqslant \frac{2^{n'}}{\binom{n}{0.01n'}} \geqslant (3/2)^{n'},$$

so $R(C) \geqslant \frac{n' \log(3/2)}{n' \log 2} \geqslant \Omega(1)$.

Thus, we can take C to be this code achieving parameters $(100 \log\log\log n, \Omega(1), \Omega(1), 2)$, and C' to be the composed Reed-Solomon code achieving parameters $(n \log n \log\log n, \Omega(1), \Omega(1), \log\log n)$, and compose them to get an $(n \log n \log\log n \log\log\log n, \Omega(1), \Omega(1), 2)$ code. Thus, we indeed get a code achieving Boolean alphabet while simultaneously having constant rate and relative distance!

So, can we just construct PCPs using these codes and be done? Well, it turns out that for the purpose of PCPs, one needs more than just constant size alphabet, constant relative distance and good rate. Indeed, in PCPs, the "proof" or "witness" that the verifier looks at will hopefully be a codeword of some code. By hopefully, we mean that this is the legitimate form of witness that we will have in mind while constructing the PCP. As is always the case though, we will need to consider any other, potentially malicious witnesses, be able to detect that they are not of the "legitimate form", and thus reject them. On top of that, our PCP verifier can only look at a few locations in the witness altogether. Thus, our code C must have the functionality that membership of a given word w in it can be tested by looking only at a few coordinates of w. This motivates the notion of local testability of codes, that we define next.

1.3 Locally Testable Codes

Informally, a locally testable code $C \subseteq \mathbb{F}_q^n$ is a code that is accompanied with a randomized tester T, which is supposed to check whether a given word $w \in \mathbb{F}_q^n$ is in C or not. Additionally, we want the tester T to be local, in the sense that it only looks at a few locations of w in order to determine if $w \in C$ or not.

Clearly, trying to distinguish between the case that $w \in C$ and $w \notin C$ using only a local tester is impossible. Indeed, if $w \notin C$ but w agrees with some codeword $c \in C$ on all but a single coordinate, the tester T would have hard time distinguishing between w and c. Thus, we relax the requirement from the tester T, and only ask it to distinguish between the case that $w \in C$, and the case that w is far from all codewords in C.

Definition 1.3. *For an error correcting code $C \subseteq \mathbb{F}_q^n$ and $h \in \mathbb{N}$, $\varepsilon, \delta > 0$ (which may be a function of the parameters of the code), an (h, ε, δ)-local tester for T is a randomized algorithm that is given an oracle access to an input $w \in \mathbb{F}_q^n$ and has the following properties:*

1. *T makes at most h oracle accesses to w.*
2. *If $w \in C$, then T accepts with probability* 1.
3. *If $\Delta(w, C) := \min_{c \in C} \Delta(w, c) \geqslant \varepsilon n$, then T rejects with probability at least δ.*

Definition 1.3 is indeed a very natural notion to consider and a lot of effort in coding theory has gone into investigating locally testable codes. For now though, it is not even clear if locally testable codes exist, and in the rest of this lecture we will see a few examples of algebraic codes that are locally testable.

1.4 Local Testability of the Reed-Solomon Code

We begin by examining the Reed-Solomon codes that we already defined, and show a "local" tester for them.[1] To motivate this tester, we begin with the following observation.

Claim 1.4. *Let $d, q \in \mathbb{N}$. For all distinct $a_0, \ldots, a_{d+1} \in \mathbb{F}_q$ there are $\alpha_0, \ldots, \alpha_{d+1} \neq 0$, such that*

1. *If $f \colon \mathbb{F}_q \to \mathbb{F}_q$ has degree at most d, then $\sum\limits_{i=0}^{d+1} \alpha_i f(a_i) = 0$.*
2. *If $f \colon \mathbb{F}_q \to \mathbb{F}_q$ is not of degree d, then for some $a_1, \ldots, a_{d+1}$ we have $\sum\limits_{i=0}^{d+1} \alpha_i f(a_i) \neq 0$.*

Proof. To get a feel for what the claim, note that if we look at $f(a_0), \ldots, f(a_d)$, then there is a unique degree d polynomial g such that $g(a_i) = f(a_i)$ (interpolation), hence if f is a degree d polynomial the value of it in a_{d+1} must be equal to $g(a_{d+1})$. In other words, d values of f determine the last one. The additional information given to us by the claim is that this deduction can be in fact be phrased as a linear equation in $f(a_0), \ldots, f(a_{d+1})$.

To establish the first item, consider the matrix $M \in \mathbb{F}_q^{(d+1)\times(d+2)}$ whose i, j entry is a_j^i. Then M has rank $d+1$, so the system of equations $M(\alpha_0, \ldots, \alpha_{d+1}) = 0$ has a non-trivial solution, and it is easily seen that these α's satisfy the first item of the claim.

[1]The reason we have put the word "local" in quotation marks is that while this local tester achieves a non-trivial testing result, it will not be useful for us in the context of PCPs.

For the second item, we prove the counter-positive. Namely, we assume that f satisfies the equality for all $a_0, \dots, a_{d+1}$, and prove that f is a polynomial of degree at most d. Take any $a_0, \dots, a_d$, and define a polynomial h of degree d such that $h(a_i) = f(a_i)$. We claim that $h \equiv f$. Indeed, taking any other a_{d+1} we find that

$$\sum_{i=0}^{d+1} \alpha_i f(a_i) = 0 = \sum_{i=0}^{d+1} \alpha_i h(a_i),$$

where the first equality is by assumption and the second equality is by the first item. Thus, simplifying we get that $h(a_{d+1}) = f(a_{d+1})$, and we are done. □

A slightly annoying feature of this claim is that the coefficients α depend on points we chose $a_0, \dots, a_{d+1}$, and circumvent that we consider a special type of $(d+2)$-tuples of points for which this is not the case, which are tuples that form an arithmetic progression.

Claim 1.5. *Let $d, q \in \mathbb{N}$. For $x, h \in \mathbb{F}_q$ we have that for $\alpha_i = \binom{d+1}{i}(-1)^i$,*

1. *If $f\colon \mathbb{F}_q \to \mathbb{F}_q$ has degree at most d, then $\sum\limits_{i=0}^{d+1} \alpha_i f(x + ih) = 0$.*

2. *If $f\colon \mathbb{F}_q \to \mathbb{F}_q$ is not of degree d, then for some x, h we have that $\sum\limits_{i=0}^{d+1} \alpha_i f(x + ih) \neq 0$.*

Proof. Essentially the same proof as in the previous section, where one checks that the solution to the linear system of equations is $\alpha_i = \binom{d+1}{i}(-1)^i$ for $i = 0, 1, \dots, d+1$. □

Given oracle accept to some $f\colon \mathbb{F}_q \to \mathbb{F}_q$ and some parameter $d \in \mathbb{N}$, Claim 1.5 suggests a local tester T: choose $x, h \in \mathbb{F}_q$ uniformly and independently, and check that $\sum\limits_{i=0}^{d+1} \alpha_i f(x + ih) = 0$.

Theorem 1.6. *The local tester T is an $(d+2, 2\delta, \delta)$ tester for $\mathsf{RS}_{d,n,q}$ for all $\delta < \frac{1}{4(d+1)^2}$.*

The rest of this section is devoted to the proof of Theorem 1.6. First, it is clear that if f is a degree d polynomial, then the tester always accepts. As for the soundness of the test, we prove it counter-positively: assuming that the tester T accepts with probability at least $1 - \delta$, we show that f is close to a degree d polynomial. For this, for each $x \in \mathbb{F}_q$ consider a random choice of $h \in \mathbb{F}_q$, and then executing the test on $x, x+h, \dots, x+(d+1)h$. Then we know that with probability $\geqslant 1 - \delta$, we have that

$$\sum_{i=1}^{d+1} \alpha_i f(x + ih) + \alpha_0 f(x) = 0,$$

so $f(x) = -\sum\limits_{i=1}^{d+1} \frac{\alpha_i}{\alpha_0} f(x + ih)$. This tells us that the value of $-\sum\limits_{i=1}^{d+1} \frac{\alpha_i}{\alpha_0} f(x + ih)$ doesn't really depend on the choice of h but rather only on x (at least with high probability), and thus one is motivated to define

$$g(x) = \mathsf{plurality}_{h \in \mathbb{F}_q} \left(-\sum_{i=1}^{d+1} \frac{\alpha_i}{\alpha_0} f(x + ih) \right).$$

First, we show that $\Delta(f,g) \leqslant 2\delta n$. Indeed, $f(x) = -\sum\limits_{i=1}^{d} \frac{\alpha_i}{\alpha_0} f(x+ih)$ with probability at least $1-\delta$, hence with probability at least $1-2\delta$ over x, we have that if we fix x, then over the choice of h we have $f(x) = -\sum\limits_{i=1}^{d+1} \frac{\alpha_i}{\alpha_0} f(x+ih)$ with probability at least $\frac{1}{2}$, in which case $f(x) = g(x)$.

Second, we show that g is a degree d polynomial. For that, we show that for each x, the plurality in the definition of x is actually achieved overwhelmingly, and more precisely that:

Claim 1.7. *For all $x \in \mathbb{F}_q$ we have*

$$\Pr_{h_1,h_2\in\mathbb{F}_q}\left[\sum_{i=1}^{d+1}\frac{\alpha_i}{\alpha_0}f(x+ih_1) = \sum_{i=1}^{d+1}\frac{\alpha_i}{\alpha_0}f(x+ih_2)\right] \geqslant 1-2(d+1)\delta.$$

Proof. Choose h_1 and h_2 randomly, and note that for each $j \neq 0$, with probability at least $1-\delta$ we have

$$\sum_{i=0}^{d+1}\alpha_i f(x+jh_2+ih_1) = 0.$$

Multiplying by α_j and summing up all of these equations for $j = 1,\dots,d+1$, we get from the Union Bound that with probability at least $1-(d+1)\delta$

$$\sum_{i=0}^{d+1}\alpha_i\sum_{j=1}^{d+1}\alpha_j f(x+jh_2+ih_1) = 0. \tag{1}$$

Similarly, for all $i \neq 0$ we have that

$$\sum_{j=0}^{d+1}\alpha_j f(x+ih_1+jh_2) = 0,$$

implying that $\sum\limits_{j=1}^{d+1}\alpha_j f(x+ih_1+jh_2) = -\alpha_0 f(x+ih_1)$ with probability at least $1-\delta$. Therefore by the Union we get that with probability at least $1-2(d+1)\delta$ we may plug this into (1) and get that

$$0 = \sum_{i=0}^{d+1}\alpha_i\sum_{j=1}^{d+1}\alpha_j f(x+jh_2+ih_1) = \alpha_0\sum_{j=1}^{d+1}\alpha_j f(x+jh_2+0\cdot h_1) + \sum_{i=1}^{d+1}\alpha_i\cdot(-\alpha_0 f(x+0\cdot h_2+ih_1)),$$

so $\sum\limits_{j=1}^{d+1}\alpha_j f(x+jh_2) = \sum\limits_{i=1}^{d+1}\alpha_i f(x+ih_1)$, finishing the proof. □

We can now show that g has degree at most d. The idea is to show that g passes all of the tests, and thereby conclude it is degree d by the second item of Claim 1.5. Take any $x,h \in \mathbb{F}_q$; we want to show that $\sum_{i=0}^{d+1}\alpha_i g(x+ih) = 0$, and to show that we will introduce two random variables h_1, h_2 that will be used to randomize the starting point of the test x and the direction of the test h. Using them, we will reduce $\sum_{i=0}^{d+1}\alpha_i g(x+ih) = 0$ to showing that a collection of $O(d^2)$ random tests of f pass.

Claim 1.8. *If $\delta < \frac{1}{4(d+1)^2}$, then g has degree at most d.*

Proof. In more detail, fix x, h and take $h_1, h_2 \in \mathbb{F}_q$ randomly; then by Claim 1.7 for each $i = 0, \ldots, d$ we have that

$$g(x+ih) = -\sum_{i_1=1}^{d+1} \frac{\alpha_{i_1}}{\alpha_0} f(x+ih+i_1h_1) \tag{2}$$

with probability at least $1-2(d+1)\delta$. Also, for all $i_1 \neq 0$ and $i \neq 0$ we have that $\sum_{i_2=0}^{d+1} \alpha_{i_2} f(x+ih+i_1h_1+i_2ih_2) = 0$, with probability at least $1-\delta$, hence by the Union Bound we get that with probability at least $1-3(d+1)\delta$ we have

$$g(x+ih) = \sum_{i_1=1}^{d+1}\sum_{i_2=1}^{d+1} \frac{\alpha_{i_1}\alpha_{i_2}}{\alpha_0^2} f(x+i(h+i_2h_2)+i_1h_1) \tag{3}$$

for all $i \neq 0$. For $i = 0$, the right hand side is

$$\sum_{i_1=1}^{d+1}\sum_{i_2=1}^{d+1} \frac{\alpha_{i_1}\alpha_{i_2}}{\alpha_0^2} f(x+i_1h_1) = -\sum_{i_2=1}^{d+1} \frac{\alpha_{i_2}}{\alpha_0} g(x) = g(x)$$

where in the first equality we used (2), and in the second one we used $\sum_{i_2=0}^{d+1} \frac{\alpha_{i_2}}{\alpha_0} = 0$ (which holds by choice of α by choosing the constant 1 function). Thus, for each i, (3) holds with probability at least $1-3(d+1)\delta$.

We expressed $g(x+ih)$ for fixed x, h as a linear combination of the values of f at points of the form $y+h'$ where y and h' are jointly uniform over $\mathbb{F}_q$. This will allow us to use the fact that the tester passes with high probability in order to analyze the sum of $g(x+ih)$ over i.

To be more specific, using the Union bound, the probability (3) holds for all $i = 0, \ldots, d+1$ is at least $1-3(d+1)^2\delta$, and multiplying by α_i and summing gives that

$$\sum_{i=0}^{d+1} \alpha_i g(x+ih) = \sum_{i_1=1}^{d+1}\sum_{i_2=1}^{d+1} \frac{\alpha_{i_1}\alpha_{i_2}}{\alpha_0^2} \sum_{i=0}^{d+1} \alpha_i f(x+i(h+i_2h_2)+i_1h_1) = 0,$$

where the last equality holds with probability $1-(d+1)^2\delta$, since it can be viewed as the test applied on $x' = x+i_1h_1$ on $h' = h+i_2h_2$, which are distributed independently in $\mathbb{F}_q$. For each $i_1, i_2 \neq 0$, the points $x+i_1h_1$ and $h+i_2h_2$ are jointly uniform from $\mathbb{F}_q$, hence $\sum_{i=0}^{d+1} \alpha_i f(x+i(h+i_2h_2)+i_1h_1) = 0$ with probability at least $1-\delta$. We get by the Union Bound that $\sum_{i=0}^{d+1} \alpha_i g(x+ih) = 0$ with probability at least $1-4(d+1)^2\delta > 0$, and since this an event that does not depend on h_1, h_2, we get that $\sum_{i=0}^{d+1} \alpha_i g(x+ih) = 0$. □

1.5 The Reed-Muller and Hadamard Codes

Returning to the discussion in the end of Section 1.2, we argued there that for the PCP application we must be using codes which are locally testable. In the previous section, we saw that Reed-Solomon codes are somewhat locally testable; here, we say "somewhat" because the locality of the tester is $\approx d$, whereas for the code to be useful at all for us, we need to take fairly large d (say $d = \Omega(n)$), which makes the local testability result we proved not very useful.

Secondly, in the end of the argument once the alphabet size of the composed code is small enough, we used a code found by a brute-force argument with constant rate, relative distance and Boolean alphabet; in the PCP application we will need to replace that code with a code that has local testability.

To resolve these issues, we will introduce two more basic families of codes: the Reed-Muller codes and the Hadamard Codes.

1.6 The Reed-Muller Codes

The Reed-Muller codes are the multi-variate analogs of the Reed-Solomon codes. Here, we again have a field $\mathbb{F}_q$, a degree parameter $d \in \mathbb{N}$, and a parameter $m \in \mathbb{N}$ which is the number of variables our polynomials have. In these notations, the Reed-Muller code is

$$\mathsf{RM}_{m,d,q} = \left\{ (f(v))_{v\in\mathbb{F}_q^m} \;\middle|\; f\colon \mathbb{F}_q^m \to \mathbb{F}_q \text{ has total degree at most } d\right\}.$$

Here, the total degree of a monomial $x_1^{i_1}\cdots x_m^{i_m}$ is $i_1 + \ldots + i_m$, and the total degree of a polynomial f is the maximum of the total degree over all monomials that appear in f. Thus, for every function $f\colon \mathbb{F}_q^m \to \mathbb{F}_q$ of the form

$$f(x_1,\ldots,x_m) = \sum_{\vec{i}: i_1+\ldots+i_m\leqslant d} c_{\vec{i}}\, x_1^{i_1}\cdots x_m^{i_m},$$

we have a corresponding codeword $(f(v))_{v\in\mathbb{F}_q^m}$ in $\mathsf{RM}_{m,d,q}$.

It is easy to show that the Reed-Muller code is a linear error correcting code, and one can analyze the parameters of it. For example, if $q \geqslant d$, then the rate of $\mathsf{RM}_{m,d,q}$ is $\frac{\binom{m+d}{m}}{q^m}$, and the relative distance is at least $1 - \frac{d}{q}$. You will establish some of its properties in the problem set.

So far, it appears that Reed-Muller codes are worse than Reed-Solomon codes, at least with respect to the rate that they offer (they do have a decent distance though if q is much larger than d). However, Reed-Muller codes have far better local testability properties compared to Reed-Solomon codes. In Reed-Solomon, to perform local testing we had to essentially read a constant fraction of the given function. In Reed-Muller, we can do much better; this will be the focus of discussion at a later point in the course, and for now we briefly explain the extra versatility afforded to us by using Reed-Muller codes.

One idea that becomes available to us when looking at multivariate polynomials, is restrictions. Indeed, suppose we have a polynomial $f\colon \mathbb{F}_q^m \to \mathbb{F}_q$ of total degree at most d, and consider a line, namely a function $\ell\colon \mathbb{F}_q \to \mathbb{F}_q^m$ of the form $\ell(t) = a + tb$ for some $a, b \in \mathbb{F}_q^m$. Then we can consider the restriction of f to the line, i.e. the function $f|_\ell(t) = f(\ell(t))$, and note that this function is a univariate polynomial in t of degree at most d. Thus, we can use local testing ideas from the Reed-Solomon code in order to perform local testing on g, and that will cost us $O(d)$ queries. Thus, it stands to reason that if we want to test if f is degree d or far from it, we could try to pick a random line ℓ, and perform the local tester of Reed-Solomon on $f|_\ell$. Indeed, there are ideas in this spirit that work (though are more difficult to prove), and we will see it in the future.

So what did we gain from this? Well, in the basic set-up of Reed-Solomon, the amount of information that a polynomial of degree d encodes is $d + 1$ symbols from $\mathbb{F}_q$ (that is, its coefficients), and we needed to invest $d + 2$ queries to locally test. In the Reed-Muller code, we could have stored much more information – roughly $\binom{m+d}{m}$ many symbols from $\mathbb{F}_q$, and the locality of the local tester does not change much. This is one of the main features that makes Reed-Muller so useful in the context of PCP.

1.7 The Hadamard Codes

The final family of codes we present is the family of Hadamard codes. This family is specified by a parameter $n \in \mathbb{N}$. For a vector $v \in \mathbb{F}_2^n$, we define $\chi_v \colon \mathbb{F}_2^n \to \mathbb{F}_2$ as $\chi_v(x) = \langle x, v\rangle$, and then define the Hadamard as

$$H_n = \left\{ (\chi_v(x))_{x\in\mathbb{F}_2^n} \,\middle|\, v \in \mathbb{F}_2^n \right\}.$$

As you will see in the problem set, H_n has relative distance $1/2$ (which is good), relative rate $\frac{n}{2^n}$ (which is not very good; we will only use it when n is already small) and alphabet size 2. One additional important feature of H_n is that it is locally testable:

Theorem 1.9. *H_n is $(3, 2\varepsilon, \varepsilon)$ locally testable for $\varepsilon < \frac{1}{8}$.*

Proof. Consider the following tester T: given oracle access to a function $f \colon \mathbb{F}_2^n \to \mathbb{F}_2$, choose $x, y \in \mathbb{F}_2^n$ uniformly and independently, query $f(x)$, $f(y)$ and $f(x+y)$ and check that $f(x+y) = f(x) + f(y)$; here, $(x+y)_i = x_i + y_i$. Then it is clear that the locality of the tester is 3, and that if $f = \chi_v$ for some $v \in \mathbb{F}_2^n$, then the tester succeeds with probability 1 as

$$\chi_v(x+y) = \langle x+y, v\rangle = \langle x, v\rangle + \langle y, v\rangle = \chi_v(x) + \chi_v(y).$$

Next, we prove that if f is 2ε far from H_n, then T rejects with probability at least ε. We prove that counter-positively; namely, assuming that T rejects with probability less than ε, we show that f is close to some χ_v. Indeed, define $g \colon \mathbb{F}_2^n \to \mathbb{F}_2$ by

$$g(x) = \mathsf{majority}_{y\in\mathbb{F}_2^n} \left(f(x+y) - f(y)\right).$$

We first claim that $\Delta(f, g) \leqslant 2\varepsilon 2^n$. Indeed, since $f(x) + f(y) = f(x+y)$ with probability at least $1 - \varepsilon$, we have that for all but $1 - 2\varepsilon$ of the x's, it holds that $\Pr_y\left[f(x) = f(x+y) - f(y)\right] \geqslant \frac{1}{2}$, in which case $f(x) = g(x)$.

Second, we claim that $g = \chi_v$ for some $v \in \mathbb{F}_2^n$, and for that we show that $g(x) + g(z) = g(x+z)$ for all $x, z \in \mathbb{F}_2^n$. Towards this end, we argue that for all x, the majority in the definition of $g(x)$ is attained:

Claim 1.10. *For all $x \in \mathbb{F}_2^n$ we have $\Pr_{y_1,y_2\in\mathbb{F}_2^n}\left[f(x+y_1) - f(y_1) = f(x+y_2) - f(y_2)\right] \geqslant 1 - 2\varepsilon$.*

Proof. With probability $1 - \varepsilon$ we have that $f(y_1) + f(y_2) = f(y_1 + y_2)$, and with probability $1 - \varepsilon$ we have that $f(x+y_1) + f(x+y_2) = f(x+y_1+x+y_2) = f(y_1+y_2)$, hence with probability $1 - 2\varepsilon$ we have that $f(x+y_1) + f(x+y_2) = f(y_1) + f(y_2)$. Since $f(y_1) + f(y_2) = f(y_1) - f(y_2)$ and $f(x+y_1) + f(x+y_2) = f(x+y_1) - f(x+y_2)$, the result follows. $\square$

The proof now quickly follows. Fix $x, z \in \mathbb{F}_2^n$ and take $y_1, y_2 \in \mathbb{F}_2^n$ randomly; then by Claim 1.10 with probability at least $1 - 6\varepsilon$ we have

$$g(x) = f(x+y_1) - f(y_1), \qquad g(z) = f(z+y_2) - f(y_2), \qquad g(x+z) = f(x+z+y_1+y_2) - f(y_1+y_2),$$

so

$$g(x) + g(z) - g(x+z) = (f(x+y_1) + f(z+y_2) - f(x+z+y_1+y_2)) - (f(y_1) + f(y_2) - f(y_1+y_2)).$$

With probability at least $1 - 2\varepsilon$ we have $f(x+y_1) + f(z+y_2) - f(x+z+y_1+y_2) = 0$ and $f(y_1) + f(y_2) - f(y_1+y_2) = 0$, hence with probability at least $1 - 8\varepsilon > 0$ we have that $g(x) + g(z) - g(x+z)$; as the last statement does not have any of the y's in it, it follows that $g(x) + g(z) - g(x+z) = 0$ for all x and z.

Taking $v \in \mathbb{F}_2^n$ by setting $v_i = g(e_i)$ where $e_i \in \mathbb{F}_2^n$ is the ith elementary basis vector, one can show that $g(x) = \chi_v(x)$, and we are done. $\square$

1.8 Composing Locally Testable Codes?

Having introduced error correcting codes, the composition technique and realizing that we need to use local testing towards PCP, we come to the question of whether our new realization combines well with the composition technique that was so crucial in construction good error correcting codes (and will be just as crucial when proving the PCP theorem).

Thinking about it schematically, suppose that we have C_1 which is an (n_1, d_1, r_1, q_1) code, and C_2 which is an (n_2, d_2, r_2, q_2) code, and C_1, C_2 are locally testable by the testers T_1 and T_2 that have locality t_1 and t_2 respectively. Is the composed code $C_1 \circ C_2$ locally testable?

Let $w \in \mathbb{F}_{q_2}^{n_1 \cdot n_2}$ be a word; how shall we go about testing it locally? A natural idea proceeds as follows: we imagine a "virtual codeword" $c_1 \in C$ from which w is derived, and then simulate the tester T_1 on it. More precisely:

1. Run the tester T_1 in order to choose locations $i_1, \ldots, i_{t_1} \in [n_1]$ to be read from the "virtual codeword".
2. Read off the blocks corresponding to these locations in w, i.e. the blocks that are supposed to have replaced the symbols in locations $i_1, \ldots, i_t$ in c_1 with codewords from C_2.
3. Decipher from the blocks the corresponding symbol in the original word from C_1 in these locations, and perform the local test T_1 on these values.

There are numerous issues with this scheme. First off, it may be the case that the blocks that we read corresponding to $i_1, \ldots, i_t$ were erroneous (after all, we care to reject words that are far from the code), in which case the deciphering in the third step would be incorrect and may even fail. In such case, the simulation of T_1 we are trying to perform on a virtual word from C_1 is incorrect.

This motivates the notion of *local decodeability*, a notion that strengthens local testing by requiring us not only to be able to locally test whether a given word is a codeword or far from the code. Instead, if w is close to a codeword c, we are required also to be able to recover any coordinate of c by performing only a few oracle access calls to w.

Another (related) issue that we point out now (but will only appear later on) is that sometimes, we will really be interested in some special coordinates i of c which are not "random looking", while only having access to a word w which is only guaranteed to be close to c. In particular, it may be the case that for this specific type of i's we always have that $w_i \neq c_i$. This motivates the idea of random self-reducibility that we have actually already seen a few times). This idea says that given a coordinate i, we can choose a few coordinates $j_1, \ldots, j_\ell$ so that marginally each one of them is distributed uniformly over $[n]$, and yet they are correlated in a way that if we know $c_{j_1}, \ldots, c_{j_\ell}$ we can also recover c_i. For example, in the Hadamard code, if we had some f that is close to χ_v, and a special point $x^\star \in \mathbb{F}_2^n$ that we really cared about and wanted to know $\chi_v(x^\star)$, we could proceed as follows. Sample $y \in \mathbb{F}_2^n$ uniformly, ask for $f(y)$ and $f(y + x^\star)$ (where we note that marginally each one of y and $y + x^\star$ is distributed uniformly over $\mathbb{F}_2^n$), and output $f(y + x^\star) - f(y)$. It is easily seen that if f and χ_v are close, then the output is $\chi_v(x^\star)$ with high probability.

2 Upcoming Lectures

Starting from the next lecture, we will see analogous ideas to the ones presented herein and are used in the algebraic proof of the PCP theorem. There are several ingredients that need to be combined well together to make the proof go through, and each one of the steps requires more time and attention.

We will first show a PCP construction achieving great completeness and soundness, albeit with a large alphabet size (analogous to the Reed-Solomon code). Following that, we will spend a few lectures on the sum-check protocol and the low-degree test (local testing for Reed-Muller), which together realize the idea of composition introduced in this lecture, and achieve an alphabet size reduction. We will need to apply this step twice, and for that we will spend a few more lectures introducing a few more ideas, most notably the block property and aggregation of queries. Finally, we will be in a position to apply the Hadamard-based PCP to finish off the proof.