18.408 Topics in Theoretical Computer Science Fall 2022
Lectures 17-20

Dor Minzer

Today we present the long-code framework in hardness of approximation, and use it to prove several tight inapproximability results. En route, we give a brief introduction to discrete Fourier analysis over the Boolean hypercube.

1 Tight Inapproximability Results: Introduction

Our primary objective in the upcoming lectures will be to prove tight inapproximability results for the 3Lin and 3SAT problems that we already saw in lecture 1. Below, we give a brief introduction to these problems.

1.1 The Complexity of Linear Equations over Finite Fields

Linear equations over fields are probably one of the most basic object in mathematics. A first course in linear algebra typically begins with a few lectures discussion how to solve a system of linear equations over a field, conditions for a solution to exist and so on. Typically, the matrix ranking algorithm (Gaussian elimination) is presented, and throughout the course many more applications of this method are presented. Thus, it makes sense to consider the complexity of solving linear systems of equations over fields from a TCS view. Since objects need to have a finite description in computer-science, it makes sense that we will discuss finite fields; otherwise, we may run into issues such as how do we even represent real-numbers, which we wish to avoid.

Given a prime power q, consider the field $\mathcal{F}_q$ and define the 3Lin_q problem as follows. An instance of the problem (X, E) consists of a set of variables $X = \{x_1, \ldots, x_n\}$ that are supposed to be assigned with values from $\mathbb{F}_q$, as well as a set E of equations. Each equation $e \in E$ is of the form $a_{1,e}x_i + a_{2,e}x_j + a_{3,e}x_k = b_e$, where $a_{1,e}, a_{2,e}, a_{3,e}, b_e$ are all field elements. Given an instance of 3Lin_q, the goal is to find an assignment $A\colon X \to \mathbb{F}_q$ that satisfies as many of the equations as possible.

Given a system (X, E) promised to be fully satisfiable, we can use our favorite Gaussian elimination algorithm to efficiently find a satisfying assignment $A\colon X \to \mathbb{F}_q$. Writing this in gap problems notations, we conclude that gap-$3\text{Lin}_q[1, 1]$ is in the class P. What happens though, if instead of promising that the system (X, E) is satisfiable, we only promise that it is$(1 - \varepsilon)$-satisfiable, where $\varepsilon > 0$ is very small? Can we efficiently find a decent assignment for the instance in this case as well?

A quick inspection of the Gaussian elimination algorithm shows that it fails miserably, so we are back to the drawing board algorithmically. A naive idea one may try is to simply choose an assignment $A\colon X \to \mathbb{F}_q$ randomly. That is, for each variable x_i in the system, choose the value of $A(x_i)$ from $\mathbb{F}_q$ uniformly; how well does this assignment perform? Fixing an equation $e \in E$ in the system, say $a_{1,e}x_i + a_{2,e}x_j + a_{3,e}x_k = b_e$, we note that if at least one of the coefficients on the left hand side are non-zero, then the distribution of $a_{1,e}A(x_i) + a_{2,e}A(x_j) + a_{3,e}A(x_k)$ is uniform over $\mathbb{F}_q$. Hence, that element will be equal to b_e with probability $1/q$. Therefore, A satisfies the equation e with probability $1/q$, so by linearity of expectation the expected number of equations that A satisfies is at least $\frac{1}{q}|E|$.

Given such "expectation guarantee", there are a few standard techniques often allow one to deduce a proper (often times, even deterministic) algorithm that achieves this expected value; you will see some of them in the problem set. For the 3Lin_q problem as above, it is indeed not hard to convert this algorithm that works "in expectation" to an algorithm that finds an assignment satisfying at least $1/q$ fraction of the equations; in any case, it follows that there is an assignment that satisfies at least $1/q$ fraction of the equations, hence gap-$3\text{Lin}_q[1-\varepsilon, 1/q]$ is also in P (regardless of what ε is).

Surely though, this naive algorithm can be improved? I doesn't even look at the system (X, E)!

Theorem 1.1. *For all prime powers q, and for all $\varepsilon, \delta > 0$, the problem gap-$3\text{Lin}_q[1-\varepsilon, 1/q+\delta]$ is NP-hard.*

In other words, the trivial algorithms above (the Gaussian elimination and the choose-a-random-assignment algorithms) are the best one can do for 3Lin_q. We will prove Theorem 1.1 in the upcoming lectures, and for simplicity we will focus on the case that $q = 2$.

1.2 The Complexity of 3SAT

The 3SAT problem is the poster NP-complete problem. Recall that a 3CNF formula consists of a set of variables $X = \{x_1, \ldots, x_n\}$ and a formula over X, $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ wherein each clause C_j is of the form $\alpha \vee \beta \vee \gamma$ where each one of α, β, γ is a literal (a variable from X or its negation).

Viewing 3SAT as an optimization problem, given a 3CNF formula $\phi(x_1, \ldots, x_n)$, the goal is to find an assignment $A\colon X \to \{0,1\}$ that satisfies as many of the clauses in ϕ as possible. In this terminology, the Cook-Levin Theorem asserts that gap-3SAT$[1,1]$ is NP-hard. Using the basic PCP theorem and some elementary ideas, one can show that there is $\varepsilon > 0$ such that gap-3SAT$[1, 1-\varepsilon]$ is NP-hard, meaning that given a satisfiable 3CNF formula ϕ, it is NP-hard to find an assignment that satisfies at least $1-\varepsilon$ of the clauses. In particular, it is NP-hard to approximate 3SAT within factor $1-\varepsilon$ for some explicit (but small) $\varepsilon > 0$. How well can one approximate 3SAT, though?

Well, we can try a random assignment idea again. Sample $A\colon X \to \{0,1\}$ by taking $A(x_i)$ to be a random bit chosen independently for each $x_i \in X$. Observe that each individual clause of the form $C = (\alpha \vee \beta \vee \gamma)$ is satisfied with probability at least $1 - 2^{-3} = 7/8$, so in expectation A satisfies at least $7/8$ of the clauses of ϕ. Using standard techniques, one can convert this guarantee to a proper, efficient algorithm that given a formula ϕ finds an assignment satisfying at least $7/8$ of its clauses. Thus, gap-3SAT$[1, 7/8]$ is in P. But surely, one can do better? The algorithm above doesn't even look at ϕ!

Theorem 1.2. *For all $\varepsilon > 0$, the problem gap-$3\text{Sat}[1, 7/8 + \varepsilon]$ is NP-hard.*

In other words, the choose-a-random-assignment algorithm is, once again, achieves the best possible approximation ratio by an efficient algorithm (assuming P≠NP). The techniques we show herein will can also be used to establish Theorem 1.2. We may prove a slightly weaker result to avoid some complications, though.

1.3 The Long-code Paradigm

In this section, we begin the discussion about the Long-code paradigm, which is a general approach for proving hardness of approximation results using the PCP theorem. To motivate the discussion, we will consider a somewhat larger class of problems that include both 3SAT and 3Lin, and discuss the steps that one often has to take in order to prove hardness results for a problem in this class.

1.3.1 Predicates and Testers that Use the Predicate

The 3SAT and the 3Lin problems are two examples of problems known as constraint satisfaction problems. There are several (non-equivalent) definitions of constraint satisfaction problem, and we present one which will help us for the purpose of this lecture. A (Boolean) constraint satisfaction problem is defined by a predicate $P\colon \{0,1\}^r \to \{0,1\}$, and an instance of it (X,E) consists of a set of variables $X = \{x_1,\ldots,x_n\}$ as well as a set of constraints E. Each constraint has the form $P(\alpha_1,\ldots,\alpha_r) = 1$, where $\alpha_1,\ldots,\alpha_r$ are literals. The goal is to find an assignment $A\colon X \to \{0,1\}$ that satisfies as many of the constraints of (X,E) as possible.

In this terminology, the 3Lin problem is the constraint satisfaction problem corresponding to the predicate P wherein $r = 3$ and $P(x,y,z) = 1$ if $x + y + z = 0 \pmod 2$. The 3SAT problem is the constraint satisfaction problem with $r = 3$ corresponding to the predicate $P(x,y,z) = (x \vee y \vee z)$.

To prove a hardness of approximation result for some predicate P, one needs to find a locally testable error correcting code, whose local tester performs checks that correspond to the predicate P. In other words, one needs to find a code $C \subseteq \{0,1\}^N$ and a local tester $\mathcal{T}$ that on input $w = (w_1,\ldots,w_N)$, samples r locations $i_1,\ldots,i_r \in [N]$ (in a randomized way) from w and then checks that $P'(w_{i_1},\ldots,w_{i_r}) = 1$. Here P' is the same as the predicate P, except that we allow to apply negations on some coordinates; for example, P' may be defined as $P'(x_1,\ldots,x_r) = P(1-x_1,x_2,\ldots,x_r)$.

The tester $\mathcal{T}$ should accept codewords from C with high probability, say c (which is typically 1 or close to it). For weak hardness results, it suffices to show that if the tester $\mathcal{T}$ accepts w with probability close to c, then w is close to a codeword. For strong hardness results, one needs to venture into the list decoding regime, and show that if the tester $\mathcal{T}$ accepts w with probability at least s (which may be much smaller than c), then w is "correlated" with some codeword from C.[1] Indeed, the quality of the eventual hardness result we will get will be s/c. Thus, the ratio between s and c that we are able to achieve determines the quality of the hardness result we prove, so we will try to get s and c to be far from each other..

Remark 1.3. *We stress that, as far as we know, finding such a code and a test is not sufficient for proving a hardness of approximation result. Indeed, in the coming lectures we will develop such code and test that have parameters that correspond to the requirement needed from Theorem 1.1, but we will need to work harder to turn this into a proof of Theorem 1.1. There is a well known conjecture in TCS, called the Unique-Games Conjecture, which if true would say that any code and test and above would yield a hardness of approximation result with matching parameters automatically, and we may discuss it later in the course.*

1.3.2 One Code for all Testers

So, does it mean that for every single new constraint satisfaction problem we face, we need to come up with a new code and a new local tester $\mathcal{T}$? For the tester $\mathcal{T}$ this is inevitable, since the tester itself has to only performs checks that correspond to the predicate we want to prove hardness for. For the code, though, there is no a-priori reason there would not be one, universal, nice enough could that would be rich enough to facilitate local testers of many different forms.

And indeeed, ideally, we would like to have a single code C that will work for all hardness results, so that at each time we only have to be concerned with designing the tester $\mathcal{T}$. Intuitively, the best chance for us to achieve such property is if the code C is maximally "redundant". Namely, the information in a codeword is so well spread so well that we can access or decode any part of it by applying any predicate on several

[1] The word correlated here appears with quotation marks since we will not actually be able to achieve correlation with a codeword, but some other notion of list-decodability that will be sufficient for our purpose.

well-chosen coordinates of it. Indeed, this is because we want to facilitate many completely different local tests (one for each predicate P), and this raises the question of what is the Boolean code C that has the most redundancies? A good candidate for such a code would be a code C of the worst possible rate (ignoring trivialities such as repetition codes). This code has a name: it is called the Long-code (the word "long" precisely describes the fact that the encoding of an element there is very, very long), and we formally define it below:

Definition 1.4 (The Long-code)**.** *Let $n \in \mathbb{N}$ and let $i \in \{1, \ldots, n\}$. The long code encoding of i is the truth table of the function $f_i \colon \{0,1\}^n \to \{0,1\}$ defined by $f_i(x) = x_i$.*

The long-code is the set $\mathsf{LC} = \left\{ (f_i(x))_{x \in \{0,1\}^n} \quad i \in [n] \right\}$.

Thus, for each coordinate $i \in [n]$, we encode i by the truth table of the dictatorship function $f_i(x) = x_i$ over $\{0,1\}^n$, which is a string of length 2^n bits. In other words, we encode a string of $\log n$ bit (the index i) by a string of length 2^n. Thus, the encoding of an index i is doubly exponential in the length of i, so indeed this code has a very bad rate hence many "redundancies". This used to be not-so-good for us early on in the course, but in the upcoming lectures it will be crucial for proving Theorem 1.1.

In the literature, local testers for the long code are often referred to as *dictatorship tests*. The reason for that comes from social choice theory. We can think of a Boolean function $f \colon \{0,1\}^n \to \{0,1\}$ as a voting scheme wherein there are n-voters. Voter i casts their vote x_i between two candidates, 0 and 1, and the function f is then applied to aggregate all of these votes: the winner of the elections is candidate $f(x_1, \ldots, x_n)$. With this in mind, the function f_i in the definition of the long-code really merits the name "dictatorship"; the outcome of the scheme would always be the opinion of the i^{th} voter, regardless of the opinions of the rest.

2 Designing a Dictatorship Test for 3Lin

In this section, we develop a dictatorship test for the 3Lin problem.

So, our goal is to query a given function $f \colon \mathbb{F}_2^n \to \mathbb{F}_2$ in 3 locations, check some linear equation on them, and say whether f is a long-code codeword or not based on it. Where do we even start? Well, this is not clear, but we have already seen something similar to that earlier in the course. More specifically, we saw a local tester for the Hadamard code over $\mathbb{F}_2$. Recall that in the Hadamard code, for each $\alpha \in \mathbb{F}_2^n$ we have a codeword, which is the truth table of $h_\alpha \colon \mathbb{F}_2^n \to \mathbb{F}_2$ defined by $h_\alpha(x) = \langle \alpha, x \rangle$. We saw a local tester for the Hadamard code, which given oracle access to a supposed codeword $f \colon \mathbb{F}_2^n \to \mathbb{F}_2$, samples $x, y \in \mathbb{F}_2^n$ uniformly, and checks that $f(x) + f(y) = f(x + y)$. We saw that if f is a Hadamard codeword, then the test passes with probability 1, and if the test passes with probability $\geqslant 1 - \varepsilon$ for $\varepsilon < 1/8$, then f is 2ε-close to a Hadamard codeword.

Note that the long-code is a sub-code of the Hadamard code; indeed, $f_i = h_\alpha$ for $\alpha = e_i$. Thus, we can use the above test to ensure that codewords will be accepted, and narrow down the class of functions that perform well in the test to functions close to h_α for some $\alpha \in \mathbb{F}_2^n$. This is a good start, but for the test to be useful for us for the purpose of Theorem 1.1, we need to improve this tester in two ways:

1. We would like to be able to argue about functions that pass the test with probability $s = 1/2 + \delta$ (as opposed to probability close to 1) because we want to get a strong hardness result for 3Lin.

2. We would like to narrow down further the functions that perform well in the test, and (roughly) only allow such functions to be h_α for α of small Hamming weight. Ideally, we would have liked to only allow α to have Hamming weight 1 (and thus be a long-code codeword), but we will not be able to do

so. Still, if we manage to guarantee α to have constantly small Hamming weight, this will correspond to at most constantly many long-code codewords.

2.1 Analyzing the Linearity Test in the List Decoding Regime

We start off by resolving the first issue, and present an analysis of the linearity tester in the small soundness regime. That is, we have a function $f\colon \{0,1\}^n \to \{0,1\}$ such that

$$\Pr_{x,y\in\mathbb{F}_2^n}[f(x)+f(y)=f(x+y)] = \frac{1}{2}+\delta, \tag{1}$$

and we would like to argue that f must have some Hadamard-ish codeword behaviour. As we saw earlier in the course, if δ is close to $1/2$, then f must be close to some Hadamard codeword h_α. In the current context, when δ is thought of as positive (but small) constant, it is natural to expect that f will be correlated with Hadamard codeword h_α. This turns out to be true, and for that we will need some basic tools from discrete Fourier analysis. [2]

Fix f as above, and let $\alpha \in \mathbb{F}_2^n$. We want to show that f and h_v are correlated, namely that for some α the number

$$c_\alpha = \Pr_{x\in\mathbb{F}_2^n}[f(x)=h_\alpha(x)] - \Pr_{x\in\mathbb{F}_2^n}[f(x)\neq h_\alpha(x)]$$

is bounded away from 0. We re-write c_α in a more suggestive form, and for that purpose we first observe that $(-1)^{f(x)+h_\alpha(x)} = 1$ if $f(x) = h_\alpha(x)$ and -1 otherwise. Therefore,

$$c_\alpha = \mathop{\mathbb{E}}_{x\in\mathbb{F}_2^n}\left[(-1)^{f(x)+h_\alpha(x)}\right] = \mathop{\mathbb{E}}_{x\in\mathbb{F}_2^n}\left[(-1)^{f(x)}(-1)^{h_\alpha(x)}\right].$$

The last expectation looks like the L_2 inner product between two functions, which are $(-1)^{f(x)}$ and $(-1)^{h_\alpha(x)}$. This suggests that it may be a good idea to define a certain vector space with the L_2 inner product on it, and use some tools from linear algebra to study it. This is indeed the case, but to make our lives (and notations) easier, it is convenient to switch to $(\{1,-1\},\cdot)$ notations as opposed to $(\{0,1\},+ \pmod 2)$ notations.

2.1.1 The Notational Switch: Going from $\{0,1\}$ to $\{1,-1\}$

Instead of working with bits $b \in \{0,1\}$, it will be more convenient for us to work with signs, $(-1)^b \in \{1,-1\}$ (thus, 1 represents 0 and -1 represents 1). Thus, instead of thinking about the function $f\colon \{0,1\}^n \to \{0,1\}$, we can think of $F\colon \{-1,1\}^n \to \{-1,1\}$ defined by $F(z) = (-1)^{f(x)}$ where $z_i = (-1)^{x_i}$ for each i. Also, instead of thinking about the function h_α, we will think of the function $\chi_\alpha\colon \{-1,1\}^n \to \{-1,1\}$, defined as $\chi_\alpha(z) = (-1)^{h_v(x)}$ where $z_i = (-1)^{x_i}$. Note that χ_α takes the form:

$$\chi_\alpha(z) = (-1)^{\langle\alpha,x\rangle} = (-1)^{\sum_{i:\alpha_i=1} x_i} = \prod_{i:\alpha_i=1}(-1)^{x_i} = \prod_{i:\alpha_i=1} z_i,$$

namely addition modulo 2 translated into multiplying signs. The function χ_α often goes by the name character (it has a special meaning when viewed as a homomorphism from $\mathbb{F}_2$ to reals with absolute value 1), and we will adopt this terminology. We note that with these notations, the parameter c_α we considered earlier takes a nice form: $c_\alpha = \mathbb{E}_{z\in\{-1,1\}^n}[F(z)\chi_\alpha(z)]$.

[2] We remark that discrete Fourier analysis is a rich enough topic to merit a separate course, so our presentation here will naturally be very partial.

2.1.2 Discrete Fourier Analysis

Now that we presented the quantity we wish to study as an inner product, we formally define the inner product space that we work with and state some basic properties of it.

Definition 2.1. *We define the inner product between real-valued functions over $\{-1,1\}^n$ as follows. For functions $F, G\colon \{-1,1\}^n \to \mathbb{R}$, define*

$$\langle F, G\rangle = \mathop{\mathbb{E}}_{z\in\{-1,1\}^n}[F(z)G(z)].$$

It is easy to check that this definition satisfies all of the properties of inner product, so now we can think of the space of real-valued functions over $\{-1,1\}^n$ as a vector space equipped with an inner product structure; we shall denote this space by $L_2(\{-1,1\}^n)$. We note that the dimension of $L_2(\{-1,1\}^n)$ is 2^n.

So why do inner products arise from considering functions f that satisfy (1)? Is there anything special about the functions χ_v with respect to this inner product that would explain the conclusion we are expecting to get? First, note that for all $\alpha \in \mathbb{F}_2^n$,

$$\|\chi_\alpha\|_2^2 = \langle \chi_\alpha, \chi_\alpha\rangle = \mathop{\mathbb{E}}_{z\in\{-1,1\}^n}[\chi_\alpha(z)\chi_\alpha(z)] = \mathop{\mathbb{E}}_{z\in\{-1,1\}^n}\left[\chi_\alpha(z)^2\right] = \mathop{\mathbb{E}}_{z\in\{-1,1\}^n}[1] = 1,$$

so each χ_α is a unit vector in this vector space. Second, note that if $\alpha = \vec{0}$, χ_α is the constant 1 function, and if $\alpha \neq \vec{0}$, then

$$\mathop{\mathbb{E}}_{z}[\chi_\alpha(z)] = \mathop{\mathbb{E}}_{z}\left[\prod_{i:\alpha_i\neq 0} z_i\right] = \prod_{i:\alpha_i\neq 0}\mathop{\mathbb{E}}_{z}[z_i] = 0,$$

so χ_α has average 0 (and hence is orthogonal to $\chi_{\vec{0}}$. Third, for $\alpha, \alpha' \in \mathbb{F}_2^n$ we have

$$\chi_\alpha(z)\chi_{\alpha'}(z) = \prod_{i:\alpha_i=1} z_i \cdot \prod_{i:\alpha_i'=1} z_i = \prod_{\substack{i \text{ such that}\\ \alpha_i=1,\alpha_i'=0 \text{ or}\\ \alpha_i=0,\alpha_i'=1}} z_i = \chi_{\alpha\oplus\alpha'}(z),$$

so if $\alpha \neq \alpha'$ then $\langle \chi_\alpha, \chi_{\alpha'}\rangle = \mathbb{E}_{z\in\{-1,1\}^n}[\chi_\alpha(z)\chi_{\alpha'}(z)] = \mathbb{E}_{z\in\{-1,1\}^n}[\chi_{\alpha\oplus\alpha'}(z)] = 0$. In other words, we have just shown the following lemma:

Lemma 2.2. *The set $\{\chi_\alpha\}_{\alpha\in\mathbb{F}_2^n}$ is an orthonormal set in $L_2(\{-1,1\}^n)$.*

With Lemma 2.2 and our earlier observation that the dimension of $L_2(\{-1,1\}^n)$ is 2^n, we get that the set $\{\chi_\alpha\}_{\alpha\in\mathbb{F}_2^n}$ is an *orthonormal basis* for $L_2(\{-1,1\}^n)$. Therefore, given any real-valued function over $\{-1,1\}^n$, say $G\colon \{-1,1\}^n \to \mathbb{R}$, we can represent G as a linear combination of χ_α:

$$G(z) = \sum_{\alpha\in\mathbb{F}_2^n} \widehat{G}(\alpha)\chi_\alpha(z).$$

The coefficients $\widehat{G}(\alpha)$ are called the Fourier coefficients of G. As χ_α is an orthonormal basis, we can say a few things about the coefficients of G:

1. **Parseval's equality:** we have a basic result known as Parseval's equality, which asserts that

$$\mathop{\mathbb{E}}_{z}\left[G(z)^2\right] = \langle G, G\rangle = \left\langle \sum_{\alpha\in\mathbb{F}_2^n} \widehat{G}(\alpha)\chi_\alpha, \sum_{\alpha'\in\mathbb{F}_2^n} \widehat{G}(\alpha')\chi_{\alpha'} \right\rangle = \sum_{\alpha,\alpha'\in\mathbb{F}_2^n} \widehat{G}(\alpha)\widehat{G}(\alpha')\left\langle \chi_\alpha, \chi_{\alpha'}\right\rangle = \sum_{\alpha\in\mathbb{F}_2^n} \widehat{G}(\alpha)^2.$$

   In particular, if G has 2-norm equal to 1 (as in our case of interest, wherein G will be ± 1 valued), then the sum of squares of Fourier coefficients of G is also 1.

2. **A formula for the Fourier coefficients:** for any $\alpha \in \mathbb{F}_2^n$ we have that

$$\langle G, \chi_\alpha\rangle = \left\langle \sum_{\alpha'\in\mathbb{F}_2^n} \widehat{G}(\alpha')\chi_{\alpha'}, \chi_\alpha \right\rangle = \sum_{\alpha'\in\mathbb{F}_2^n} \widehat{G}(\alpha')\left\langle \chi_{\alpha'}, \chi_\alpha\right\rangle = \widehat{G}(\alpha).$$

   Hence, a Fourier coefficient $\widehat{G}(\alpha)$ is the inner product of G with the corresponding basis function χ_α.

In particular, from the last remark it follows that the parameters c_α that we defined earlier are none other than the Fourier coefficients of the function F! It therefore makes sense that the above inner product perspective will be useful for us to understand functions satisfying (1) (provided that, indeed, our guess that it implies correlation with a Hadamard codeword is indeed correct). But how do we do that?

Well, the first step is to phrase (1) in terms of the function F instead of f, and a quick inspection shows that it is equivalent to

$$\Pr_{x,y\in\{-1,1\}}[F(x)F(y) = F(xy)] \geqslant \frac{1}{2} + \delta, \tag{2}$$

where $(xy)_i = x_i y_i$. Still, it is not clear how to apply our inner-product Fourier machinery, and we need to arithmetize this probability statement into an expectation statement. Note that given (2), we get that

$$\Pr_{x,y\in\{-1,1\}}[F(x)F(y) \neq F(xy)] \leqslant \frac{1}{2} - \delta,$$

so

$$\mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n}[F(x)F(y)F(xy)] = \Pr_{x,y\in\{-1,1\}}[F(x)F(y) = F(xy)] - \Pr_{x,y\in\{-1,1\}}[F(x)F(y) \neq F(xy)] \geqslant 2\delta.$$

We can now try to plug in our Fourier expansion for F and hope for the best. Indeed, we write

$$F(x) = \sum_{\alpha\in\mathbb{F}_2^n} \widehat{F}(\alpha)\chi_\alpha(x), \qquad F(y) = \sum_{\beta\in\mathbb{F}_2^n} \widehat{F}(\beta)\chi_\beta(y), \qquad F(xy) = \sum_{\gamma\in\mathbb{F}_2^n} \widehat{F}(\gamma)\chi_\gamma(xy),$$

and note that $\chi_\gamma(xy) = \chi_\gamma(x)\chi_\gamma(y)$. We get that

$$\begin{aligned}
\mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n}[F(x)F(y)F(xy)] &= \mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n}\left[\sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \widehat{F}(\alpha)\widehat{F}(\beta)\widehat{F}(\gamma)\chi_\alpha(x)\chi_\beta(y)\chi_\gamma(x)\chi_\gamma(y)\right] \\
&= \mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n}\left[\sum_{\alpha,\beta,w\in\mathbb{F}_2^n} \widehat{F}(\alpha)\widehat{F}(\beta)\widehat{F}(\gamma)\chi_{\alpha\oplus\gamma}(x)\chi_{\beta\oplus\gamma}(y)\right] \\
&= \sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \widehat{F}(\alpha)\widehat{F}(\beta)\widehat{F}(\gamma) \mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n}[\chi_{\alpha\oplus\gamma}(x)\chi_{\beta\oplus\gamma}(y)] \\
&= \sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \widehat{F}(\alpha)\widehat{F}(\beta)\widehat{F}(w) \mathop{\mathbb{E}}_{x\in\{-1,1\}^n}[\chi_{\alpha\oplus\gamma}(x)] \mathop{\mathbb{E}}_{y\in\{-1,1\}^n}[\chi_{\beta\oplus\gamma}(y)].
\end{aligned}$$

Note that unless $\alpha = \beta = \gamma$ the corresponding summand is 0, hence we get that

$$2\delta \leqslant \mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n}[F(x)F(y)F(xy)] = \sum_{\alpha\in\mathbb{F}_2^n} \widehat{F}(\alpha)^3.$$

By Parseval's equality, the squares of $\widehat{F}(\alpha)$ sum up to 1, so if they were all very small, the sum of third powers would be even smaller. This says that at least one of these numbers is large; indeed:

$$2\delta \leqslant \sum_{\alpha\in\mathbb{F}_2^n} \widehat{F}(\alpha)^3 \leqslant \max_\alpha \widehat{F}(\alpha) \cdot \sum_{\alpha\in\mathbb{F}_2^n} \widehat{F}(\alpha)^2 = \max_\alpha \widehat{F}(\alpha).$$

Thus, we have just proved the following theorem.

Theorem 2.3. *Suppose $F\colon \{-1,1\}^n \to \{-1,1\}$ satisfies that $\Pr_{x,y\in\{-1,1\}}[F(x)F(y) = F(xy)] \geqslant \frac{1}{2} + \delta$. Then there exists $\alpha \in \mathbb{F}_2^n$ such that $\widehat{F}(\alpha) \geqslant 2\delta$.*

Tracing back, we see that

$$\Pr_x[f(x) = h_\alpha(x)] = \frac{1}{2}(1 + c_\alpha) = \frac{1}{2}(1 + \widehat{F}(\alpha)) \geqslant \frac{1}{2} + \delta,$$

so Theorem 2.3 gives a positive answer to our guess, and indeed any f satisfying (1) must be correlated with a Hadamard codeword. We are therefore done arguing about the first item in Section 2.

In the future, we may need an additional property regarding Fourier coefficients that is related to this test. In words, Theorem 2.3 guarantees that under the condition specified therein, there is a Fourier character on which F has a significant coefficient. Can there be many such Fourier coefficients?

Lemma 2.4. *Suppose $F\colon \{-1,1\}^n \to \{-1,1\}$ is any function, and let $\varepsilon > 0$. Then the number of α's such that $\widehat{F}(\alpha) \geqslant \varepsilon$ is at most $1/\varepsilon^2$.*

Proof. Denoting the number of these v's by k, we note that by Parseval's equality,

$$1 = \mathop{\mathbb{E}}_z\left[F(z)^2\right] = \sum_\alpha \widehat{F}(\alpha)^2 \geqslant k\varepsilon^2,$$

and re-arranging gives $k \leqslant 1/\varepsilon^2$. □

Next, we shift our attention to the second item in Section 2.

2.2 The Noisy Linearity Test

Theorem 2.3 gives us a tester for the Hadamard code which almost fits the conditions stated in 1.3. The main difference is that in the soundness, we managed to show correlation with a Hadamard codeword, as opposed to correlation with a long-code codeword. As discussed earlier, we will not be able to fully resolve this issue under the standard notion of what "correlation" is; there are less standard notions which we will be able to achieve. For the moment, we will adapt an ad hoc ad approach, but remark that the more principled notions are concerned with parameters called "influences" and "low-degree influences" of a function; we may discuss them at a later point in the course.

So, how do we turn Theorem 2.3 into a result that at least somewhat resembles correlation with a long code word? Well, morally speaking, our goal here is to distinguish between functions of the form χ_v for v which has large cardinality (we will want to penalize them so that the test rejects them often), and functions χ_v for which the cardinality of v is small.

2.2.1 Applying Noise

To motivate our approach consider the $\alpha = e_1$, so that $\chi_\alpha(z) = z_1$, and $\alpha = e_1 + \ldots + er$, where r is a large constant, so that $\chi_\alpha(z) = z_1 \cdots z_r$. Suppose we have an input $z \in \{-1,1\}^n$, and we lightly perturb it to arrive at an input $z' \in \{-1,1\}^n$. By that, we mean that we take an ε-biased distributed $a \in \{-1,1\}^n$, meaning that $a_i = 1$ with probability $1-\varepsilon$ and otherwise $a_i = -1$, and define $z' = az$. What can we say about $\chi_\alpha(z)$ and $\chi_\alpha(z')$ in the two cases above?

1. If $\alpha = e_1$, then $\chi_\alpha(z') = a_1 z_1 = a_1 \chi_v(z)$, so $\chi_\alpha(z') = \chi_v(z)$ with probability $1-\varepsilon$. Thus, the values of the function on these two points are very correlated.

2. If $\alpha = e_1 + \ldots + e_r$, then $\chi_\alpha(z') = \chi_\alpha(z) \prod_{i=1}^{r} a_i$. Looking at $\prod_{i=1}^{r} a_i$ and thinking of r as a large constant, the distribution of the products seems to be close to uniform in $\{-1,1\}$. Indeed, if $r > 1/\varepsilon$ we expect there would be few -1's among $a_1, \ldots, a_r$, and if r is much larger than that we expect that the probability there would be an odd number of -1's to be roughly $1/2$. This turns out to be true, hence if $r > r_0(\varepsilon)$, the values $\chi_\alpha(z')$ and $\chi_\alpha(z)$ are barely correlated.

The above observation points out to a property that distinguishes Hadamard codewords corresponding to v of small support size (which we think of as long-code words), and Hadamard codewords corresponding to v of large support size. This property is called noise sensitivity.

Upon seeing this, a natural test to consider would be: (1) apply the linearity test, that is, choose $x, y \in \{-1,1\}^n$ and check that $F(x)F(y) = F(xy)$, and (2) apply the noise test, that is, choose $a \in \{-1,1\}^n$ which is ε-biased, and check that $F(xa) = F(x)$. This combined test can be shown to work (well, almost), in the sense that long-code codewords pass it with probability $1-\varepsilon$, and if the test passes with probability $1/2 + \varepsilon$ on F, then F is correlated with χ_v for v of small cardinality.

The primary issue with this test is that it is no longer of a 3Lin form. We are making an AND of two linear equations, which is no longer a linear equation. Hence we cannot use it to prove hardness for 3Lin. To resolve this issue, we need to incorporate the noise test into the linearity test.

2.2.2 Incorporating the Noise Test into the Linearity Test

We now present a single test that combines the linearity test and the noise test together. The test picks $x, y \in \{-1,1\}^n$ uniformly and independently, and $a \in \{-1,1\}^n$ according to the ε-biased distribution independently, and checks that $F(axy) = F(x)F(y)$. We call this the noisy linearity test, and with respect to it we have the following theorem:

Theorem 2.5. *Let $F\colon \{-1,1\}^n \to \{-1,1\}$ be a function, and $\varepsilon > 0$.*

1. *If F is a long-code codeword, that is, $F(x) = x_i$ for some $i \in [n]$, then the noisy linearity tester passes with probability $1-\varepsilon$.*

2. *If F passes the noisy linearity tester with probability $1/2 + \delta$, then there is $\alpha \in \mathbb{F}_2^n$ such that*

$$(1-2\varepsilon)^{|\alpha|} \widehat{F}(\alpha) \geqslant 2\delta.$$

In particular, there is α of size at most $\frac{\ln(1/\delta)}{2\varepsilon}$ such that $\widehat{F}(\alpha) \geqslant 2\delta$.

Proof. For the first item, suppose that $F(x) = x_i$. Note that the test passes if and only if $a_i = 1$, which happens with probability $1 - \varepsilon$, and the first item is proved.

For the second item, note that under the assumption we have that

$$\mathop{\mathbb{E}}_{x,y,a}[F(axy)F(x)F(y)] \geqslant 2\delta.$$

We now use the Fourier expansion of F as in Theorem 2.3. We have

$$\begin{aligned}\mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n,a}[F(x)F(y)F(axy)] &= \mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n,a}\left[\sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n}\widehat{F}(\alpha)\widehat{F}(\beta)\widehat{F}(\gamma)\chi_\alpha(x)\chi_\beta(y)\chi_\gamma(x)\chi_w(y)\chi_\gamma(a)\right] \\ &= \sum_{\alpha,\beta,w\in\mathbb{F}_2^n}\widehat{F}(\alpha)\widehat{F}(\beta)\widehat{F}(\gamma)\mathop{\mathbb{E}}_x[\chi_{\alpha\oplus\gamma}(x)]\mathop{\mathbb{E}}_y[\chi_{\beta\oplus\gamma}(y)]\mathop{\mathbb{E}}_a[\chi_\gamma(a)].\end{aligned}$$

As before, unless $\alpha = \beta = \gamma$ the corresponding summand is 0. As for the expectation over a, we have

$$\mathop{\mathbb{E}}_a[\chi_\gamma(a)] = \mathop{\mathbb{E}}_a\left[\prod_{i:\gamma_i=1}a_i\right] = \prod_{i:\gamma_i=1}\mathop{\mathbb{E}}_a[a_i] = \prod_{i:\gamma_i=1}(1-2\varepsilon) = (1-2\varepsilon)^{|\gamma|}.$$

Combining, we get that

$$2\delta \leqslant \mathop{\mathbb{E}}_{x,y\in\{-1,1\}^n,a}[F(x)F(y)F(axy)] \leqslant \sum_{\alpha\in\mathbb{F}_2^n}(1-2\varepsilon)^{|\alpha|}\widehat{F}(\alpha)^3 \leqslant \max_\alpha(1-2\varepsilon)^{|\alpha|}\widehat{F}(\alpha)\sum_{\alpha\in\mathbb{F}_2^n}\widehat{F}(\alpha)^2,$$

and the proof is concluded by appealing to Parseval's equality to say that $\sum_{\alpha\in\mathbb{F}_2^n}\widehat{F}(\alpha)^2 = 1$. □

Theorem 2.5 is an important milestone towards proving the hardness of approximation of 3Lin, and provided a very strong form of the PCP theorem called Unique-Games PCPs (which is conjectured to exists but not known), it implies Theorem 1.1 almost immediately.

To see a proper NP-hardness result though we will need to work harder. This will also allow us to see where the structure of the constraints in our PCP theorem enters the picture at all.

3 Utilizing the Long-code Encoding in PCPs

3.1 Local Testing and Verifying Constraints via the Long-code

Having designed a dictatorship tester that fits the mold in Section 1.3, we now try to understand how this is all related to getting a hardness of approximation result for 3Lin using the PCP theorem. Our intention is to use the ideas above to carry out a reduction from the label cover problem to the 3Lin problem, so let us think of a label cover instance $\Psi = (L \cup R, E, \Sigma_L, \Sigma_R, \{\Phi_e\}_{e\in E})$. Therein, we are supposed to assign a label from Σ_L to each vertex $u \in L$, and a label from Σ_R to each vertex from $v \in R$. In our reduction, we will attempt to encode the labels for the vertices of Ψ using the long-code encodings, and then use the ideas above to verify that the encodings that we get are indeed long-code encodings. We will also have to be able to verify, using these encodings, constraints on the edges of Ψ.

To be more specific, fix a vertex $u \in L$ and suppose we want to assign u the label $\sigma_u \in \Sigma_L$. Thinking of "$[n]$" in the discussion so far as Σ_L, we will want to encode that label via its corresponding long code

codeword, that is, by the function $f_u \colon \{-1,1\}^{\Sigma_L} \to \{-1,1\}$ defined as $f_u(z) = z_{\sigma_u}$. Similarly, for $v \in R$, we want to encode the fact that v is supposed to get the label $\sigma_v \in \Sigma_L$ via the long-code codeword corresponding to the function $f_v \colon \{-1,1\}^{\Sigma_R} \to \{-1,1\}$ defined as $f_v(x) = x_{\sigma_v}$. As usual in PCP though — and as we have seen several times by now — while when thinking about the reduction we have the legitimate honest encodings in mind, for the reduction to be sound we must design a tester that indeed makes sure that the encodings are as we expect them to be. Hence, to facilitate the soundness of such reduction we need to:

1. Ensure that given functions $g_u \colon \{-1,1\}^{\Sigma_L} \to \{-1,1\}$, $g_v \colon \{-1,1\}^{\Sigma_L} \to \{-1,1\}$ are indeed valid long-code encodings of some labels for u and v. We will not be able to guarantee such as strong property — even in the 99% regime we were only able to guarantee closeness to a legal codeword, and we are now aiming for a low-soundness result. We did see some result in the course in this regime – namely the plane versus line test – and here we expect to get a similar-in-spirit list decoding type statement, saying that we can associate g_u and g_v with a short list of possible labels.

2. Check the constraint between u and v using the encodings g_u and g_v.

Addressing each one of these issues separately is not too difficult using the ideas we've seen so far. Indeed, for the first item we can run the linearity+noise tester on each one of g_u and g_v separately. The second issue is also not very difficult to handle, and in a sense we have seen something along these lines happening when we applied the quadratic Hadamard code and used it to check quadratic equations in the encoded vector. Still, the details in the case of the long-code are somewhat different, and we discuss them next.

Verifying the constraints. The second item requires a bit more thought; recall that the constraint between u and v is a projection constraint, meaning that it is defined by a map $\phi_{u,v} \colon \Sigma_L \to \Sigma_R$. Thus, given a point $x \in \{-1,1\}^{\Sigma_R}$, we can define the pull-back point $y = \phi_{u,v}^{-1}(x)$ defined by $y_i = x_{\phi_{u,v}(i)}$ for each $i \in \Sigma_L$. In words, for each label $\sigma \in \Sigma_R$, in y all coordinates corresponding to the pre-images of σ under $\phi_{u,v}$ have the value x_σ. We note that if f_u and f_v are legitimate long code encodings of labels σ_u and σ_v, then

$$f_u(y) = y_{\sigma_u} = x_{\phi_{u,v}(\sigma_u)}, \qquad f_v(x) = x_{\sigma_v},$$

so if (σ_u, σ_v) satisfy the constraint on $e = (u,v)$, that is, if $\phi_{u,v}(\sigma_u) = \sigma_v$, then $f_u(y) = f_v(x)$ for every choice of x. If, on the other hand, (σ_u, σ_v) do not satisfy the constraint on $e = (u,v)$, then choosing x uniformly, the values of $x_{\phi_{u,v}(\sigma_u)}$ and x_{σ_v} are independent, and hence are equal with probability $1/2$.

It follows that to address the second item, we may hope to test g_u and g_v by taking $x \in \{-1,1\}^{\Sigma_R}$ uniformly, setting $y = \phi_{u,v}^{-1}(x)$ and checking that $g_u(y) = g_v(x)$. This almost works; the vigilant reader may notice that the issue is that while the distribution of x is uniform, the distribution of y is not. In fact, the number of y's that are possible to get in this way is negligible. Therefore, since we expect g_u only to be correlated with a long-code codeword, it may well be the case there will be errors on all of these points y, making this test meaningless.

Fortunately, this is an issue we have already encountered and we know how to solve. We can use random self-correction: we will take $x \in \{-1,1\}^{\Sigma_R}$ uniformly, $y = \phi_{u,v}^{-1}(x)$ and $z \in \{-1,1\}^{\Sigma_L}$ uniformly and then test that $g_u(zy)g_u(z) = g_v(x)$. The idea is that if g_u is a long-code codeword, then $g_u(zy)g_u(z) = g_u(y)$ hence the test remains the same for legitimate encodings. For the soundness now, each one of the points z and zy is distributed uniformly in $\{-1,1\}^{\Sigma_L}$, so a malicious adversary cannot corrupt all of these locations together.

3.2 Combining the Tests into a Single Test

Everything we said so far makes sense, but there is one significant issue that we have to deal with. The tester above check that an AND of 3 linear equations, which is not a linear equation. We could use such testers to prove hardness results for soundness close to 1, but since we are shooting for soundness which is close to $1/2$, this is not good enough.

Fortunately, there is a magical way of combining all of the above testers. Namely, we can design a test that checks a single linear equation which incorporates together the linearity+noise tests for both g_u and g_v, as well as the constraint verification test. Here it is:

1. Sample $x \in \{-1,1\}^{\Sigma_R}$ uniformly and set $y = \phi_{u,v}^{-1}(x)$.
2. Sample $z \in \{-1,1\}^{\Sigma_L}$ uniformly and $a \in \{-1,1\}^{\Sigma_L}$ according to the ε-biased measure. Namely, for each $i \in \Sigma_L$ independently, set $a_i = 1$ with probability $1-\varepsilon$, and $a_i = -1$ otherwise.
3. Test that $g_u(zya)g_u(z) = g_v(x)$.

We analyze the test in the following theorem, and to state it we need to introduce some notation. For a character $\alpha \in \mathbb{F}_2^{\Sigma_L}$, we define $\beta = \pi_{u,v}^{\mathsf{odd}}(\alpha) \in \mathbb{F}_2^{\Sigma_R}$ as the vector β in which $\beta_j = 1$ if and only if the number of preimages i of j under $\phi_{u,v}$ such that $\alpha_i = 1$ is odd. In other words, $\beta_j = \sum_{i:\phi_{u,v}(i)=j} \alpha_i$.

Theorem 3.1. *Suppose that $g_u \colon \{-1,1\}^{\Sigma_L} \to \{-1,1\}$ and $g_v \colon \{-1,1\}^{\Sigma_R} \to \{-1,1\}$ are functions such that $\Pr_{x,y,z,a}[g_u(zya)g_u(z) = g_v(x)] \geqslant \frac{1}{2} + \delta$. Then*

$$\sum_{|\alpha| \leqslant \frac{\ln(1/\delta)}{\varepsilon}} \widehat{g_u}(\alpha)^2 \widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 \geqslant \delta^2.$$

Proof. From the premise of Theorem 3.1, we have that

$$\mathop{\mathbb{E}}_{x,y,z,a}[g_u(zya)g_u(z)g_v(x)] \geqslant 2\delta,$$

and we next use Fourier analysis to analyze the left hand side. We have

$$\begin{aligned}
\mathop{\mathbb{E}}_{x,y,z,a}[g_u(zya)g_u(z)g_v(x)] &= \mathop{\mathbb{E}}_{x,y,z,a}\left[\sum_{\alpha \in \mathbb{F}_2^{\Sigma_L}} \widehat{g_u}(\alpha)\chi_\alpha(zya) \sum_{\gamma \in \mathbb{F}_2^{\Sigma_L}} \widehat{g_u}(\gamma)\chi_\gamma(z) \sum_{\beta \in \mathbb{F}_2^{\Sigma_R}} \widehat{g_v}(\beta)\chi_\beta(x)\right] \\
&= \sum_{\substack{\alpha,\gamma \in \mathbb{F}_2^{\Sigma_L} \\ \beta \in \mathbb{F}_2^{\Sigma_R}}} \widehat{g_u}(\alpha)\widehat{g_u}(\gamma)\widehat{g_v}(\beta) \mathop{\mathbb{E}}_{x,y,z,a}[\chi_\alpha(zya)\chi_\gamma(z)\chi_\beta(x)],
\end{aligned}$$

and we analyze the inner expectation. By the multiplicativity of characters, we get that

$$\mathop{\mathbb{E}}_{x,y,z,a}[\chi_\alpha(zya)\chi_\gamma(z)\chi_\beta(x)] = \mathop{\mathbb{E}}_{a}[\chi_\alpha(a)] \mathop{\mathbb{E}}_{z}[\chi_{\alpha\oplus\gamma}(z)] \mathop{\mathbb{E}}_{x}\left[\chi_\beta(x)\chi_\alpha(\phi_{u,v}^{-1}(x))\right].$$

The first expectation is equal to $(1-2\varepsilon)^{|\alpha|}$, the second expectation is non-zero if and only if $\alpha = \gamma$. As for the third expectation, we note that

$$\chi_\alpha(\phi_{u,v}^{-1}(x)) = \prod_{i:\alpha_i=1} \phi_{u,v}^{-1}(x)_i = \prod_{i:\alpha_i=1} x_{\phi_{u,v}(i)} = \chi_{\phi_{u,v}^{\mathsf{odd}}(\alpha)}(x),$$

so the last expectation is equal to $\mathbb{E}_x\left[\chi_{\beta\oplus\phi_{u,v}^{\mathsf{odd}}(\alpha)}(x)\right]$, hence it is non-zero if and only if $\beta=\phi_{u,v}^{\mathsf{odd}}(\alpha)$. Thus, we get that

$$2\delta \leqslant \mathop{\mathbb{E}}_{x,y,z,a}[g_u(zya)g_u(z)g_v(x)] = \sum_{\alpha\in\mathbb{F}_2^{\Sigma_L}} (1-2\varepsilon)^{|\alpha|}\widehat{g_u}(\alpha)^2\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha)).$$

In the previous argument, at this stage we simply pulled out one of the Fourier coefficients outside and used Parseval's inequality to bound, but this time a bit more care is needed. Taking square and using Cauchy-Schwarz, we get that

$$\begin{aligned}(2\delta)^2 &\leqslant \left(\sum_{\alpha\in\mathbb{F}_2^{\Sigma_L}} \widehat{g_u}(\alpha)\cdot(1-2\varepsilon)^{|\alpha|}\widehat{g_u}(\alpha)\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))\right)^2\\ &\leqslant \sum_{\alpha\in\mathbb{F}_2^{\Sigma_L}} \widehat{g_u}(\alpha)^2\cdot\sum_{\alpha\in\mathbb{F}_2^{\Sigma_L}}(1-2\varepsilon)^{2|\alpha|}\widehat{g_u}(\alpha)^2\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2\\ &= \sum_{\alpha\in\mathbb{F}_2^{\Sigma_L}}(1-2\varepsilon)^{2|\alpha|}\widehat{g_u}(\alpha)^2\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2.\end{aligned}$$

Note that the contribution from α such that $|\alpha|\geqslant \ln(1/\delta)/\varepsilon$ is at most

$$(1-2\varepsilon)^{\frac{2\ln(1/\delta)}{\varepsilon}}\sum_{\alpha\in\mathbb{F}_2^{\Sigma_L}}\widehat{g_u}(\alpha)^2\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 \leqslant e^{-4\ln(1/\delta)}=\delta^4,$$

so we conclude that

$$\sum_{|\alpha|\leqslant\frac{\ln(1/\delta)}{\varepsilon}}\widehat{g_u}(\alpha)^2\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 \geqslant \sum_{|\alpha|\leqslant\frac{\ln(1/\delta)}{\varepsilon}}(1-2\varepsilon)^{2|\alpha|}\widehat{g_u}(\alpha)^2\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 \geqslant \delta^2. \qquad \square$$

3.3 What Does Theorem 3.1 Even Mean?

Ideally, instead of Theorem 3.1 we would have liked to say that there is an α with small cardinality such that the product $|\widehat{g_u}(\alpha)|\ \widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))$ is significant. This means that each one of the Fourier coefficients $|\widehat{g_u}(\alpha)|$ and $\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))$ is significant. We encourage the reader to indeed think of Theorem 3.1 for now in this way, and next explain how one may go about using such statement.

Note that if both α and $\phi_{u,v}^{\mathsf{odd}}(\alpha)$ are not the all 0 vectors, then the supports of α and $\phi_{u,v}^{\mathsf{odd}}(\alpha)$ contain pairs of labels σ_u and σ_v that satisfy the constraint $\phi_{u,v}$. Indeed, to see that take any $\sigma_v\in\mathsf{supp}({\phi_{u,v}}^{\mathsf{odd}}(\alpha))$ and any pre-image of it σ_u from the support of α (there exists such one by the definition of $\phi_{u,v}^{\mathsf{odd}}$). Thus, looking at the Fourier coefficients of g_u gives us potential way of choosing labels for u: look at all of the α's of small cardinality such that $|\widehat{g_u}(\alpha)|$ is significant, choose one of them, and choose the label σ_u of u to be random from the support of α. This is indeed going to be the idea, however the execution will be a bit different so as to work with the conclusion we get from Theorem 3.1.

There is one issue though: what if the α that we chose is $\vec{0}$? In that case, we would not be able to get any label out of it. We will ensure that this never happens by forcing the Fourier coefficients of our functions g_u and g_v corresponding to $\vec{0}$ to always be 0. Note that

$$\widehat{g_u}(\vec{0}) = \mathop{\mathbb{E}}_z[g_u(z)],$$

so we will want to force g_u has expectation 0. We will do that by forcing g_u to be an *odd function*, meaning that $g(-z) = -g(z)$. We note that dictatorships (which are legitimate assignments) are odd functions, so this will not hurt our completeness. To achieve this, we will use a technique called *folding*.

4 NP-hardness of 3Lin: the Proof of Theorem 1.1

In this section, we combine the tools we developed over the last few lectures to prove Theorem 1.1. We do so by reducing from the PCP theorem proved in previous lectures, which we restate below for convenience.

Theorem 4.1. *For all $\eta > 0$, there is $k \in \mathbb{N}$ such that the problem gap-Label-Cover$[1, \eta]$ is NP-hard on instances with alphabet size at most k and bi-regular constraint graphs.*

In the next section, we give a formal description of the reduction from label cover to 3Lin. Following that in subsequent sections, we analyze the completeness and the soundness of the reduction. For convenience, we will reduce the label cover problem to weighted version of the 3Lin problem, in which each equation is assigned a weight, and instead of counting the fraction of equations that are satisfied, we count the total weight of the equations that are satisfied.[3]

4.1 The Reduction

Given a label cover instance $\Psi = (G = (L \cup R, E), \Sigma_L, \Sigma_R, \Phi = (\phi_e)_{e \in E})$, we construct a weighted 3Lin instance (X, E, w) as follows.

The variables of the system. For every vertex $u \in L$ and a location in its long-code $z \in \{-1, 1\}^{\Sigma_L}$ we create a variable $g_u(z)$. For every vertex $v \in R$ and a location in its long-code $x \in \{-1, 1\}^{\Sigma_R}$, we create a variable $g_v(x)$.

The equations of the system. The equations and the weights of them are defined according to the follows randomized process:

1. Choose an edge $(u, v) \in E$ uniformly.
2. Take $x \in \{-1, 1\}^{\Sigma_R}$ uniformly and $z \in \{-1, 1\}^{\Sigma_R}$ uniformly. Let $y = \phi_{u,v}^{-1}(x)$.
3. Take $a \in \{-1, 1\}^{\Sigma_R}$ according to the ε-biased distribution, that is, $a_i = 1$ with probability $1 - \varepsilon$ and otherwise $a_i = -1$.
4. Create the equation $g_u(ayz)g_u(z)g_v(x) = 1$.

Folding the variables. Recall that for each u, viewing the assignment to the variables $g_u(z)$ as a function over z, we wanted to ensure that g_u is an odd function. We can do this by having a variables only for z such that $z_1 = 1$. Thus, in each equation in which a variable $g_u(z)$ appeared wherein $z_1 = -1$, we can replace it by $-g_u(-z)$. Thus, the number of variables in our system is in fact smaller and an assignment to them only specifies the values of the function g_u on half of the points in $\{-1, 1\}^{\Sigma_L}$. There is a unique way though to

[3] As in the case of the Set-cover problem, there are standard techniques that can be used to convert this result into a NP-hardness result for unweighted instances.

complete these values to an odd function, and this is the function that we will have in mind as specified by an assignment of values to the constructed system of linear equations.

This completes the description of the reduction, and we next analyze it.

4.2 The Completeness of the Reduction

Lemma 4.2. *If Ψ is satisfiable, then there is an assignment to (X, E, w) that satisfies at least $1 - \varepsilon$ fraction of the equations.*

Proof. Suppose that Ψ is satisfiable, and let A_L and A_R be satisfying assignments. We define an assignment to the variables of the equations by giving to $g_u(z)$ the value $z_{A_L(u)}$, and giving to $g_v(x)$ the value $x_{A_R(u)}$. A randomly chosen equation from the system takes the form $g_u(ayz)g_u(z)g_v(x) = 1$ for $(u, v) \in E$ chosen uniformly and x, y, z, a as above, which using our chosen assignment this equation amounts to

$$(ayz)_{A_L(u)} z_{A_L(u)} x_{A_R(v)} = 1.$$

Note that the left hand side is equal to

$$a_{A_L(u)} y_{A_L(u)} z_{A_L(u)} z_{A_L(u)} x_{A_R(v)} = a_{A_L(u)} y_{A_L(u)} x_{A_R(v)} = a_{A_L(u)} x_{\phi_{u,v}(A_L(u))} x_{A_R(v)} = a_{A_L(u)},$$

where in the last transition we used the fact that A_L and A_R satisfy all constraints, and in particular the constraint on $e = (u, v)$, so $\phi_{u,v}(A_L(u)) = A_R(v)$. Thus, as the probability that $a_{A_L(u)} = 1$ is $1 - \varepsilon$, we get that our assignment satisfies $1 - \varepsilon$ fraction of the equations. □

4.3 The Soundness of the Reduction

Fix $\varepsilon > 0$. We show that for every $\delta > 0$, for sufficiently small $\eta > 0$, if at least $\frac{1}{2} + \delta$ of the equations can be satisfied, then we can find an assignment to Ψ satisfying at least $\delta' > 0$ fraction of the constraints. Formally:

Lemma 4.3. *For all $\varepsilon, \delta > 0$ there is $\delta' > 0$ such that if there is an assignment to (X, E, w) satisfying at least $1/2 + \delta$ of the equations, then $\mathsf{val}(\Psi) \geqslant \delta'$.*

Thus, taking η in the PCP to be smaller than δ', we conclude that if $\mathsf{val}(\Psi) < \eta$, then at most $\frac{1}{2} + \delta$ of the equations in the system can be satisfied, which gives the soundness of the construction.

The rest of this section is devoted to the proof of Lemma 4.3. For an edge $e = (u, v) \in E$, denote

$$\delta_{u,v} = \mathop{\mathbb{E}}_{x,z,a} [g_u(ayz)g_u(z)g_v(x)].$$

Note that the fraction of equations that are satisfied is $\mathbb{E}_{(u,v)\in E} [(1 + \delta_{u,v})/2]$, so by the assumption we get that $\mathbb{E}_{(u,v)\in E} [\delta_{u,v}] \geqslant 2\delta$. Thus, defining

$$E' = \{ e \in E \mid \delta_e \geqslant \delta \},$$

and noting that $\delta_{u,v} \leqslant 1$ always, we get by an averaging argument that E' contains at least δ fraction of the edges in E. We next describe a probabilistic assignment to Ψ, and show that it satisfies each edge in $e \in E'$ with significant probability.

4.3.1 Defining the Probabilistic Assignment

Next, we describe a probabilistic assignment to the vertices of Ψ based on the functions g_u and g_v.

The assignment to the left vertices. We define the assignment for $u \in L$ as follows. Looking at the values of the variables $g_u(z)$ as values of a Boolean function g_u, we note that by Parseval's equality that $\sum_{\alpha} \widehat{g_u}(\alpha)^2 = \mathbb{E}_z\left[g_u(z)^2\right] = 1$, so $\widehat{g_u}(\alpha)^2$ is a distribution over characters. Thus, to choose the label of u, we choose $\alpha \in \mathbb{F}_2^{\Sigma_L}$ with probability $\widehat{g_u}(\alpha)^2$, and then choose $A_L(u)$ uniformly from the support of α.

The assignment to the right vertices. We define the assignment for a vertex $v \in R$ in a similar way. We choose $\beta \in \mathbb{F}_2^{\Sigma_L}$ with probability $\widehat{g_v}(\beta)^2$, and then choose $A_R(v)$ uniformly from the support of β.

4.3.2 Analyzing the Probabilistic Assignments

We next analyze the performance of the probabilistic assignment. Fix $e = (u, v) \in E'$, so that we know that $\mathbb{E}_{x,z,a}\left[g_u(ayz)g_u(z)g_v(x)\right] \geqslant \delta$. Hence, from Theorem 3.1 we get that

$$\sum_{|\alpha| \leqslant \frac{\log(1/\delta)}{\varepsilon}} \widehat{g_u}(\alpha)^2 \widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 \geqslant \delta^2,$$

and we next relate the left hand side to the probability that the probabilistic assignment satisfies e. Inspecting, the term $\widehat{g_u}(\alpha)^2\widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2$ corresponds to the probability that u chose the character α and v chose the character $\beta = \phi_{u,v}^{\mathsf{odd}}(\alpha)$. Note that by folding, we have ensured that α nor β can be $\vec{0}$ (otherwise the probability would be 0), hence there are pairs of labels in the supports of α and β that satisfy the constraint between u and v. Thus, conditioned on u and v picking the characters α and β, the probability that they choose a pair of labels that satisfy $\phi_{u,v}$ is at least $\frac{1}{|\alpha||\beta|}$. Hence, the probability that the probabilistic assignment above satisfies e is at least

$$\begin{aligned}
\sum_{\alpha} \frac{1}{|\alpha|\ \ \phi_{u,v}^{\mathsf{odd}}(\alpha)} \widehat{g_u}(\alpha)^2 \widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 &\geqslant \sum_{|\alpha| \leqslant \frac{\log(1/\delta)}{\varepsilon}} \frac{1}{|\alpha|^2} \widehat{g_u}(\alpha)^2 \widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 \\
&\geqslant \frac{\varepsilon^2}{\log(1/\delta)^2} \sum_{|\alpha| \leqslant \frac{\log(1/\delta)}{\varepsilon}} \widehat{g_u}(\alpha)^2 \widehat{g_v}(\phi_{u,v}^{\mathsf{odd}}(\alpha))^2 \\
&\geqslant \frac{\varepsilon^2}{\log(1/\delta)^2} \delta^2.
\end{aligned}$$

Therefore, letting W_e be the event that the probabilistic assignment satisfies the edge e, we have that $\sum_{e \in E} W_e$ represents the number of constraints in Ψ that are satisfied, and

$$\mathbb{E}\left[\sum_{e \in E} W_e\right] \geqslant \sum_{e \in E'} \mathbb{E}\left[W_e\right] \geqslant \sum_{e \in E'} \frac{\varepsilon^2}{\log(1/\delta)^2}\delta^2 = \frac{\varepsilon^2}{\log(1/\delta)^2}\delta^2 \ \ E' \ \ \geqslant \frac{\varepsilon^2}{\log(1/\delta)^2}\delta^3 \left|E\right|.$$

In particular, there is an assignment to Ψ satisfying at least $\delta' = \frac{\varepsilon^2}{\log(1/\delta)^2}\delta^3$ of the constraints.

18.408 Topics in Theoretical Computer Science: Probabilistically Checkable Proofs
Fall 2022