CS 761 : Derandomization and Pseudorandomness

Amit Rajaraman

Last updated October 27, 2022

Contents

§1. Introduction

1.1. Lecture 1: Matrix multiplication

We begin with a question.

Problem. Given three $n \times n$ matrices A, B, C, decide whether $AB = C$.

One naïve way to do this is to compute AB and check if it is identical to C. The naïve implementation of this runs in $O(n^3)$, while the best known implementation at the time runs in about $O(n^{2.373\cdots})$.
Can we do the required in $O(n^2)$ time, perhaps in a random fashion (with some probability of failure)?

Consider the following algorithm to start with. For each row in C, choose an entry randomly and verify that it matches the corresponding entry in AB. In a similar spirit, a second algorithm is to choose n entries of C randomly and verify.

If $AB = C$, it is clear that no matter how we choose to test, we shall return that the two are indeed equal. The probability we would like to minimize is

$$\Pr[\text{the algorithm outputs yes} \mid AB \neq C].$$

Of course, this probability depends on A, B, C. This probability is over the randomness inherent in the algorithm, not in some choosing of A, B, C.

When AB and C differ at only one entry, the earlier proposed algorithm has a success probability of $1/n$ (so the quantity mentioned above is $1 - 1/n$). This is very bad, as it means that to reduce the failure probability to some constant, we would need to repeat this n times.

An algorithm that does the job is as follows.
Randomly choose $r \in \{0,1\}^n$. Compute ABr and Cr, and verify that the two are equal. This is an $O(n^2)$ algorithm, since multiplying a matrix with a vector takes $O(n^2)$ and we perform this operation thrice, in addition to an $O(n)$ verification step at the end.

We claim that the failure probability of this algorithm is at most $1/2$.
The failure probability can be rephrased as follows. Let $x, y \in \mathbb{R}^{1\times n}$. What is $\Pr[xr = yr \mid x \neq y]$? The earlier failure probability is at most equal to this, with equality attained (in a sense) when the two matrices differ at exactly one row.
This in turn is equivalent to the following. Let $z \in \mathbb{R}^{1\times n}$. What is $\Pr[zr = 0 \mid z \neq 0]$? Suppose that $z_i \neq 0$ for some i. For any choice of the remaining $n-1$ bits, at most one of the two options for the ith bit can result in $zr = 0$.
Let us do this slightly more formally. Assume wlog that $z_n \neq 0$. Then,

$$\begin{aligned} \Pr\left[z_1 r_1 + \cdots + z_n r_n = 0 \mid z_n \neq 0\right] &= \Pr\left[r_n = -\frac{z_1 r_1 + \cdots + z_{n-1} r_{n-1}}{z_n} \mid z_n \neq 0\right] \\ &\leq \max_{r_1,\ldots,r_{n-1}} \Pr\left[r_n = -\frac{z_1 r_1 + \cdots + z_{n-1} r_{n-1}}{z_n} \mid z_n \neq 0, r_1, \ldots, r_{n-1}\right] \end{aligned}$$

which is plainly at most $1/2$ – we cannot have that both 0 and 1 are equal to the quantity of interest!

Remark. If we instead choose r from $\{0, 1, \ldots, q-1\}^n$ instead, the failure probability now goes down at most $1/q$. There is a tradeoff at play here between the reduction in the failure probability and the increase in the number of random bits (it goes from n to $O(n \log q)$).

Question. Can we reduce the number of random bits in this algorithm? Can we make it deterministic?

To answer the question of determinism, suppose the algorithm designer chooses k vectors $r^{(1)}, \ldots, r^{(k)} \in \mathbb{R}^n$ and tests whether $ABr^{(i)} = Cr^{(i)}$. This will fail if $k < n$. Indeed, an adversarial input is a z that is nonzero but with $zr^{(i)} = 0$ for $1 \leq i \leq k$.

The determinism here is in the sense that the vectors are chosen before the inputs are provided.

On the other hand, we *can* reduce the number of random bits used. In fact, we can go to about $O(\log n)$ random bits. The goal of derandomization is to use a smaller number of random bits (perhaps by conditioning together previously independent bits), without losing the power of the earlier independent bits.
Let

$$A(x) = a_0 + a_1 x + \cdots + a_d x^d$$

be a nonzero polynomial of degree d. Choose x randomly from $\{0, 1, \dots, q-1\}$. It is not difficult to see that

$$\Pr_{x \sim \{0,1,\dots,q-1\}} \left[A(x) = 0\right] \leq \frac{d}{q}.$$

Inspired by this, we can reduce randomness as follows. Choose x randomly from $\{0, 1, \dots, 2n-1\}$, and set $r = (1, x, x^2, \dots, x^{n-1})$. Then,

$$\Pr[z_1 r_1 + z_2 r_2 + \cdots + z_n r_n = 0] = \Pr\left[z_1 + z_2 x + z_2 x^2 + \cdots + z_n x^n\right] \leq \frac{n-1}{2n-1} \leq \frac{1}{2}.$$

There are some other issues that enter the picture here, namely the bit complexity now that x^{n-1} has $O(n)$ bits. One easy fix for this is to perform all operations modulo some prime.

1.2. Lectures 3–4: Pairwise independence

1.2.1. Lecture 3

Let $X_1, \dots, X_n$ be random variables such that for any distinct i, j, X_i, X_j are independent:

$$\Pr[X_i = \alpha, X_j = \beta] = \Pr[X_i = \alpha] \Pr[X_j = \beta].$$

This is referred to as *pairwise independence*. Analogously, we can define *k-wise independence*, which requires that any subset of at most k random variables is independent.

Example 1. Let random variables X_1, X_2 take values in $\{0, 1\}$ uniformly, and let $X_3 = X_1 \oplus X_2$. This set of random variables is pairwise independent, but not completely independent!

Given a cut $(S, \overline{S})$ of a graph, denote

$$\partial S = \{(u, v) : u \in S, v \notin S\}.$$

Consider an algorithm that chooses a uniformly random cut S of the vertex set V (which corresponds to independently choosing each vertex with probability $1/2$). Then,

$$\mathbb{E}[|\partial S|] = \sum_{e \in E} \Pr[e \in \partial S] = \sum_{\{u,v\} \in E} \Pr[u \in S, v \notin S] + \Pr[u \notin S, v \in S] = |E|/2.$$

In particular, this gives (in expectation) a $1/2$-approximation of a max-cut.[1]

Now, note that this algorithm does not require independence of all the $|V|$ vertex-choosings, it suffices to have pairwise independence! This begs the question, how do we generate n pairwise independent while using a small number of actual random bits?
Bouncing off the idea in the previous example, we can take k random bits $X_1, \dots, X_k$, and generate $2^k - 1$ pairwise independent random bits by considering $\bigoplus_{i \in S} X_i$ for each non-empty $S \subseteq [k]$ (why are these pairwise independent?).
Consequently, we can generate n pairwise random bits using just $O(\log(n))$ random bits.

[1]Using Markov's inequality, it gives a $1/2$-approximation with probability at least $1/2$.

Remark. Since we have just $\log n$ random bits, we can cycle through all the possible choices for the bits, since there are only n choices! This gives a deterministic polynomial time $1/2$-approximation algorithm for the max-cut problem. Instead of looking at all the $O(2^n)$ cuts, it is enough to look at $O(n)$ cuts.
Interestingly, this does not even look at the structure of the graph!

Proposition 1.1. To generate n pairwise independent random bits, we require $\Omega(\log n)$ independent random bits.

Proof. Suppose that given k independent random bits $Y_1, \dots, Y_k$, we can come up with n pairwise independent random bits $X_1, \dots, X_n$. Let $f_i : \{0,1\}^k \to \{0,1\}$ for $1 \leq i \leq n$ be defined by $X_i = f_i(Y_1, \dots, Y_k)$. Also, denote $f_i^{-1}(1) = \{x \in \{0,1\}^k : f_i(x) = 1\}$.
The basic constraint that $\Pr[X_i = 1] = 1/2$ means that $|f_i^{-1}(1)| = 2^{k-1}$ and the pairwise independence constraint gives that for distinct i, j, $|f_i^{-1}(1) \cap f_j^{-1}(1)| = 2^{k-2}$. Let M be the $n \times 2^k$ matrix such that $M_{ij} = f_i(j)$ (in the sense of the binary expansion of j).
The previous constraints then just say that $MM^\top = 2^{k-2}(I + J)$, where J is the all ones matrix.
Note that the $n \times n$ matrix $2^{k-2}(I + J)$ is of rank n. It follows that $\operatorname{rank}(M) = \operatorname{rank}(MM^\top) = n$, so $2^k \geq n$ and we are done! ■

Alternatively, after getting M, one may observe that if we replace 0 with -1, then the rows of M are orthogonal, which again gives the required.
Now, what happens if we want to generate pairwise independent functions instead of just bits? Can we do better? In particular, can we generate pairwise independent random variables $X_1, \dots, X_n$ that uniformly take values in $\mathbb{F}_q$, where q is a prime power?

One simple construction is similar to the earlier one – take $k := \log n$ random values $y_1, \dots, y_k$ from $\mathbb{F}_p$, and consider $\sum_{i \in S} y_i$ for each non-empty $S \subseteq [k]$. This takes $\log n \cdot \log |\mathbb{F}|$ random bits.

A better construction for $n = q$ is as follows – randomly choose $a_0, a_1 \in \mathbb{F}$, and let the required random variables be $\{a_1 z + a_0 : z \in \mathbb{F}_q\}$. This takes just $\log n + \log |\mathbb{F}|$ bits! We leave the details of checking this to the reader.

1.2.2. Lecture 4

In the above construction for generating q pairwise independent random variables uniform in $\mathbb{F}_q$, if we set $q = 2^r$, then this in fact generates q pairwise independent random bits $\log q$ times, using only $2 \log q$ independent random bits!
The naïve method to do this would involve generating q pairwise independent random bits $\log q$ times, which takes $(\log q)^2$ bits.

Further, we can generalize the construction to n of the form q^r by considering $\{a_0 + \sum_{i=1}^r a_i x_i : x_i \in \mathbb{F}_q\}$, where the a_i are iid drawn from $\mathbb{F}_q$.
This idea can further be generalized to k-wise independence as well, taking a degree-$(k-1)$ polynomial $\{\sum_{i=0}^{k-1} a_i x^i : x \in \mathbb{F}_q\}$ instead. Why are these k-wise independent? Fix distinct $x_1, x_2, \dots, x_k \in \mathbb{F}_q$ and $\alpha_1, \dots, \alpha_k \in \mathbb{F}$. Is it true that

$$\Pr\left[\sum_j a_j x_i^j = \alpha_i \text{ for all } i\right] = \frac{1}{q^k}?$$

Indeed, there is a unique solution $(a_0, \dots, a_{k-1})$ to this since the matrix corresponding to the system of equations is a Vandermonde matrix, which has nonzero determinant (even over $\mathbb{F}_q$).

Exercise 1.1. Show that a Vandermonde matrix is invertible.

Solution

Suppose instead that there is a nonzero vector v such that $Mv = 0$, where M is our $k \times k$ Vandermonde matrix of interest. This gives a nonzero polynomial of degree at most $k-1$ with k roots, which is not possible.

1.3. Lectures 4–5: Counting distinct elements in a stream

1.3.1. Lecture 4 (continued)

Pseudorandomness has various applications in streaming algorithms. We generally have storage space that is far smaller than the input. We also have only one "pass" at the input and cannot look at older input. We can however run multiple copies of the same algorithm as we get the input, and in this case this can give better results.

Problem. Suppose we are getting a stream of items $a_1, \dots, a_m$ in $[n]$. Count the number of distinct elements that appear.

A realistic example of the above is trying to find the number of unique visitors to a website.
One trivial way to do this is to store an array of size n of all the elements seen so far (or perhaps marking the elements which have been seen). This requires $O(n)$ space.
Can we go to $O(\log n)$ space, perhaps slightly giving up precision?

Let h be a function that maps each element in $[n]$ to $[0,1]$ (the continuous interval) uniformly randomly. That is, each $h(i)$ is independently uniformly randomly distributed in $[0,1]$. We start with a variable m set at ∞. For a new a in the stream, we set $m \leftarrow \min(m, h(a))$. Finally, output $1/m - 1$.
The random variable m is essentially the minimum of k random variables iid drawn from $[0,1]$, where k is the number of unique elements. Then, $\mathbb{E}[m] = 1/(k+1)$.

1.3.2. Lecture 5

Before moving on, let us verify that $\mathbb{E}[m] = 1/(k+1)$? We have that for $x \in [0,1]$,

$$\Pr[m \geq x] = \Pr[h(i) \geq x \text{ for all } i] = (1-x)^k.$$

Therefore,

$$\mathbb{E}[m] = \int_0^1 x \cdot k(1-x)^{k-1}\,\mathrm{d}x = \int_0^1 n(x^{k-1} - x^k)\,\mathrm{d}x = \frac{1}{k+1}.$$

Now, we still have to store all n outputs of h, so this has not really introduced any lower storage space. Choose a field $\mathbb{F}$ with $|\mathbb{F}| = N \geq n$. We shall choose $h(i)$ from $\mathbb{F}$ (or rather, $[N]$) such that they are pairwise independent. Recall that we had seen how to do this in Lectures 3 and 4. This construction only requires us to store the a and b from the algorithm, and we can compute $h(i) = ai + b$ whenever needed. This also lowers the space requirement to $O(\log n)$.
We shall now output $(N/m) - 1$ instead of $(1/m) - 1$.
We want to show that $(N/m) - 1$ is "close" to k with high probability. That is, let us try to bound

$$\Pr\left[(1-\epsilon)\frac{N}{k} \leq m \leq (1+\epsilon)\frac{N}{k}\right]$$

from below.
Define

$$Y_{i,\lambda} = \mathbb{1}_{h(i)>\lambda} = \begin{cases} 1, & \text{if } h(i) > \lambda \\ 0, & \text{otherwise.} \end{cases}$$

Also define

$$Y_\lambda = \sum_{i \in S} Y_{i,\lambda},$$

where S is the set of the k distinct elements that are seen. Then, we want to find

$$\Pr\left[Y_{(1-\epsilon)\frac{N}{k+1}} = 0 \text{ and } Y_{(1+\epsilon)\frac{N}{k+1}} \neq 0\right].$$

Indeed, m is at least the lower bound iff no element in the stream is mapped to something less than it, and at most the upper bound iff at least one element is mapped to something less than it. Now,

$$\mathbb{E}[Y_\lambda] = \sum_{i \in S} \mathbb{E}[Y_{i,\lambda}] \approx k\lambda/N.$$

Then, using Markov's inequality,

$$\Pr[Y_\lambda \geq 1] \leq \mathbb{E}[Y_i] = \frac{k\lambda}{N}$$

and as a result,

$$\Pr[Y_{(1-\epsilon)\frac{N}{k}} = 0] = \Pr\left[m \geq (1-\epsilon)\frac{N}{k}\right] \geq \epsilon.$$

Observe that thus far, we have not used any sort of independence.

Lemma 1.2. If $X_1, \ldots, X_n$ are pairwise independent real-valued random variables,

$$\operatorname{Var}\left[\sum_i X_i\right] = \sum_i \operatorname{Var}[X_i].$$

Proof. We have

$$\begin{aligned}
\operatorname{Var}\left[\sum_i X_i\right] &= \mathbb{E}\left[\left(\sum_i X_i - \mathbb{E}[X_i]\right)^2\right] \\
&= \mathbb{E}\left[\sum_i (X_i - \mathbb{E}[X_i])^2 + 2\sum_{i<j}(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\right] \\
&= \sum_i \mathbb{E}\left[(X_i - \mathbb{E}[X_i])^2\right] + 2\sum_{i<j}\mathbb{E}\left[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\right] \\
&= \sum_i \operatorname{Var}[X_i] + 2\sum_{i<j}\mathbb{E}[X_i - \mathbb{E}[X_i]]\mathbb{E}[X_j - \mathbb{E}[X_j]]. && (X_i, X_j \text{ are independent})
\end{aligned}$$

■

Now, set $U = (1+\epsilon)N/k$, so we have $\mathbb{E}[Y_U] = 1 + \epsilon$. By the above lemma,

$$\operatorname{Var}[Y_U] = k\operatorname{Var}[Y_{i,U}] = k \cdot \frac{U}{N}\left(1 - \frac{U}{N}\right) = (1+\epsilon)\left(1 - \frac{1+\epsilon}{k}\right).$$

Therefore, using Chebyshev's inequality,

$$\begin{aligned}
\Pr[Y_U \neq 0] &\geq 1 - \Pr\left[|Y_U - (1+\epsilon)| \geq (1+\epsilon)\right] \\
&\geq 1 - \frac{(1+\epsilon)\left(1 - \frac{1+\epsilon}{k}\right)}{(1+\epsilon)^2} \geq 1 - \frac{1}{1+\epsilon} = \frac{\epsilon}{1+\epsilon}.
\end{aligned}$$

Finally,

$$\begin{aligned}
\Pr\left[Y_{(1-\epsilon)\frac{N}{k}} = 0 \text{ and } Y_{(1+\epsilon)\frac{N}{k}} \neq 0\right] &\geq 1 - \left(\Pr\left[Y_{(1-\epsilon)\frac{N}{k}} \neq 0\right] + \Pr\left[Y_{(1+\epsilon)\frac{N}{k}} \neq 0\right]\right) \\
&\geq \epsilon + \frac{\epsilon}{1+\epsilon} - 1.
\end{aligned}$$

§2. Expander graphs and applications

2.1. Lectures 6–7: Magical graphs and two applications

2.1.1. Lecture 6

Expander graphs are interesting because they are "pseudorandom" – they behave like random objects.
We recall the subject of error correcting codes, pioneered by Shannon in 1948. It studies the idea of introducing "redundancy" when transmitting messages so that the messages are understandable even in the presence of errors.

Definition 2.1. A *code* $\mathcal{C}$ is a subset of $\{0,1\}^n$. The elements of a code are called *codewords*.

Definition 2.2. Given $x, y \in \{0,1\}^n$, the *Hamming distance* $d_H(x,y)$ between x and y is $|\{i \in [n] : x_i \neq y_i\}|$ and the *relative distance* $\Delta(x,y)$ between x and y is $d_H(x,y)/n$. The distance $d_H(\mathcal{C})$ of a code $\mathcal{C}$ is $\min_{\substack{x,y\in\mathcal{C}\\ x\neq y}} d_H(x,y)$.

The idea of this is that given a word in $\{0,1\}^k$, we translate it bijectively into a codeword in $\{0,1\}^n$ and transmit it. Upon receiving the message, we decode the received word in some way to get a word.
One simple way is to decode a received word as the codeword closest to it, in the sense of the Hamming distance. This scheme allows the correction of errors if the received word is at Hamming distance less than $(1/2)d_H(\mathcal{C})$ from the transmitted word.

Definition 2.3. The *rate* of a code is defined by

$$\operatorname{Rate}(\mathcal{C}) = \frac{\log|\mathcal{C}|}{n}.$$

We also define the *relative distance*

$$\delta(\mathcal{C}) = \frac{d_H(\mathcal{C})}{n}.$$

One question that should immediately come to mind is: given a relative distance, what is the minimum rate required to achieve it? In less formal terms, what is the minimum amount of redundancy needed? We state it more formally.

Problem. Given constants $\delta_0, r_0 \in (0,1)$, when can we construct codes $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ such that $\delta(\mathcal{C}_n) \to \delta_0$ and $\operatorname{Rate}(\mathcal{C}_n) \to r_0$?

This also presents another follow-up question: if codes of the above form exist, do there exist efficient encoding and decoding algorithms for the code? We do not look at this
Consider another question.

Problem. Suppose we have an algorithm $\mathcal{A}$ with "one-sided error". This means that if x is in the language L of interest, $\mathcal{A}(x)$ is yes with probability 1, but if x is not in the language L, $\mathcal{A}(x)$ is no with probability $\frac{15}{16}$.
How would one go about making the error probability very small, without using too many random bits?

One simple idea which we have discusses is to repeat the experiment a large number of times and output no if we get a no at any point. Indeed, if we repeat it ℓ times, the error probability goes down to $\leq (1/16)^\ell$.
However, the fault with this is that if the algorithm uses k independent random bits (say), then repeating it ℓ times requires ℓk independent random bits! Could we make it $\ell + k$? It turns out that this *is* possible.

The two questions we have described seem incredibly different, but the answers to both are yes, with the ideas behind both involving "expander graphs". Before getting to this, we define something else.

Definition 2.4 (Magical graphs)**.** A bipartite graph $G = (L \sqcup R, E)$ is said to be (n,m,d)*-magical*, $m \geq (3n/4)$, if

1. $|L| = n$, $|R| = m$,

2. for any $v \in L$, $\deg(v) = d$, and
3. for every subset $S \subseteq L$ with $|S| \leq n/10d$, $|\Gamma(S)| \geq (5d/8)|S|$.

Above, $\Gamma(S)$ denotes the neighbourhood of S.
Typically, n and m are of similar orders and d is a constant. This says that any "small" subset expands a lot – the neighbours of the vertices in the subset do not coincide too much. Ideally, with no intersection between neighbourhoods, we would have $|\Gamma(S)| = d|S|$, and we are demanding about half of this.
First, we shall see why magical graphs exist. Following this, we connect them to the questions we looked at earlier.

Theorem 2.5. For $d \geq 24$ and sufficiently large n, (n, m, d)-magical graphs exist.

Proof. For each vertex in L, choose its d neighbours randomly. Let $S \subseteq L$ with $|S| = s \leq n/10d$ and $T = R$ with $|T| = (5d/8)s$,

$$\Pr\left[\Gamma(S) \subseteq T\right] \leq \left(\frac{|T|}{m}\right)^{ds} \leq \left(\frac{5ds}{8m}\right)^{ds}.$$

This is for a *fixed* S, T however. Using the union bound,

$$\begin{aligned}
\Pr\left[\exists S, T \text{ as above such that } \Gamma(S) \subseteq T\right] &\leq \sum_{S,T} \left(\frac{5ds}{8m}\right)^{ds} \\
&\leq \sum_{s=1}^{n/10d} \binom{n}{s} \binom{m}{5ds/8} \left(\frac{5ds}{8m}\right)^{ds} \\
&\leq \sum_{s=1}^{n/10d} \left(\frac{ne}{s}\right)^{s} \left(\frac{8me}{5ds}\right)^{5ds/8} \left(\frac{5ds}{8m}\right)^{ds} && \left(\binom{n}{k} \geq (ne/k)^k\right) \\
&= \sum_{s=1}^{n/10d} \left(\frac{ne}{s}\right)^{s} e^{5ds/8} \left(\frac{5ds}{8m}\right)^{3ds/8} \\
&\leq \sum_{s=1}^{n/10d} \left(\frac{ne}{s}\right)^{s} e^{5ds/8} \left(\frac{5ds}{6n}\right)^{3ds/8} && (m \geq 3n/4) \\
&= \sum_{s=1}^{n/10d} \left(\frac{s}{n}\right)^{s(3d/8-1)} e^{s(5d/8+1)} \left(5d/6\right)^{3ds/8} \\
&\leq \sum_{s=1}^{n/10d} (10d)^{-s(3d/8-1)} e^{s(5d/8+1)} \left(5d/6\right)^{3ds/8} && (s/n \leq 1/10d) \\
&\leq \sum_{s=1}^{\infty} (10d)^{-s(3d/8-1)} e^{s(5d/8+1)} \left(5d/6\right)^{3ds/8} \\
&= \frac{\alpha}{1-\alpha},
\end{aligned}$$

where $\alpha = (10d)^{1-(3d/8)} e^{(5d/8)+1} (5d/6)^{3d/8}$. The above is less than 1 when $\alpha < 1/2$. To check for what values of d

this is true,

$$\begin{aligned}\log\alpha &= \left(1-\frac{3d}{8}\right)(\log 10+\log d)+1+\frac{5d}{8}+\frac{3d}{8}\left(\log(5/6)+\log d\right)\\ &= \log d + d\left(\frac{5}{8}-\frac{3}{8}\log(10)+\frac{3}{8}\log(5/6)\right)+(1+\log 10)\\ \frac{\mathrm{d}\log\alpha}{\mathrm{d}d} &\approx \frac{1}{d}-0.306,\end{aligned}$$

which is negative for $1/d < 0.306$ (or equivalently, $d \geq 5$). Since α is decreasing in d for $d \geq 24$, it suffices to check that $\alpha < 1/2$ when $d = 24$. Indeed, it is easily verified that $\alpha \approx 0.413 < 1/2$ in this case, completing the proof. ■

Now, let us look at reduction of randomness using magical graphs.
Let $\mathcal{A}$ be an algorithm that uses k random bits with error probability $< 1/16$. Take $n = 2^k$, and let $G = (L \sqcup R, E)$ be a (n, n, d)-magical graph. Choose a random vertex $v \in L$, and take all d neighbours $u_1, \ldots, u_d$ of v. Each u_i can be thought of as a k bit string. For $i = 1, \ldots, d$, run $\mathcal{A}$ with u_i as the choice of random bits. Observe that we are only using k random bits here, namely in the choice of v.

Why does the error probability go down?
Let $B \subseteq \{0,1\}^k$ be the set of "bad" inputs for algorithm $\mathcal{A}$. We know that $|B| \leq n/16$. What is the probability of failure when we run it d times as described above? The algorithm fails iff every u_i is in B.
We claim that there are less than $n/10d$ such vertices v with every neighbour in B. Suppose instead that there are is a set S with $n/10d$ vertices with all neighbours in B. Then,

$$\frac{n}{16} > |B| \geq \Gamma(S) \geq \frac{5d}{8}|S| \geq \frac{n}{16},$$

which is a contradiction.
Therefore, the probability of failure is at most $1/10d$.
We can make d very large (up to a limit forced by $n = 2^k$), so the probability of failure can be made very small. Using the same number of random bits, we have managed to significantly decrease the error probability.
The issue now however is that the above scheme requires the construction of exponentially large magical graphs. We require a very efficient algorithm (polynomial in $\log n$) to sample the neighbours of a random vertex in a magical graph.

2.1.2. Lecture 7

Lemma 2.6. Given a (n, m, d)-magical graph, for any $S \subseteq L$ of size at most $n/10d$, there exists $v \in \Gamma(S)$ such that v has a unique neighbour in S.

Proof. Suppose instead that no such v exists. Then,

$$d|S| = |e(\Gamma(S), S)| \geq 2|\Gamma(S)| \geq 2 \cdot \frac{5d}{8}|S|,$$

a contradiction. ■

Consider some $(n, m = 3n/4, d)$-magical graph, and let M be the $m \times n$ adjacency matrix of the graph, where M_{ij} is 1 iff there is an edge between the ith vertex on the right and the jth vertex on the left, and 0 otherwise.

Definition 2.7. A code $\mathcal{C} \subseteq \{0,1\}^n$ is said to be a *linear code* if it is a subspace when viewed as a subset of the vector space $\mathbb{F}_2^n$.

This is equivalent to saying that if $x, y \in \mathcal{C}$, then $x + y \in \mathcal{C}$. Observe that the distance of a linear code is equal to the minimum weight of its codeword.

Consider the code defined by

$$\mathcal{C} = \{x : Mx = 0\},$$

the null space of M (over $\mathbb{F}_2$). In coding theory lingo, one would say that M is the parity check matrix of $\mathcal{C}$. Evidently,

$$|\mathcal{C}| = 2^{n-\operatorname{rank}(M)} \text{ and } \operatorname{Rate}(\mathcal{C}) = \frac{n - \operatorname{rank}(M)}{n} \geq \frac{n-m}{n} \geq \frac{1}{4}.$$

We claim that $d_H(C) > n/10d$, so the relative distance is at least $1/10d$. Suppose instead that there is some $x \in \mathcal{C}$ such that $\operatorname{wt}(x) \leq n/10d$. Let $S \subseteq [n]$ be the subset of L such that $v \in S$ iff $x_v \neq 0$. By Lemma 2.6, there exists some $u \in R$ such that $M_{uv} = 1$ for precisely one $v \in S$. However, this implies that $(Mx)_u = \sum_v M_{uv} x_v = 1$.

2.2. Lectures 7, 8, 10: Testing connectivity of undirected graphs

2.2.1. Lecture 7 (continued)

Problem. Given a graph G and two vertices $s, t \in V(G)$, determine if there is a path between s and t.

To test connectivity of a graph, we can just test the above by iterating through all $t \in V(G)$. We work in the setting where running time is not an issue (as long as it is polynomial), but we have space constraints.

One way to do this, of course, is through a depth-first/breadth-first search. This requires $\Omega(n)$ space however, which is more than we can afford.

Recall that there is a path between s, t iff $((I + A)^n)_{st} \neq 0$. Indeed, $(A^i)_{st}$ gives the number of length i walks from s to t, and $(I + A)^n$ is just some weighted of sum of the A^i. Can we compute $(I + A)^n$ is $O(\log^2 n)$ space, say? We remark that here, space means that the size of the input and output tapes are "large", but we cannot alter the input tape and once we write something to the output tape, we cannot change it; we have an intermediate tape of "small" size which is what we use for computation.
The answer is yes, but we do not say why.

Consider the following algorithm, that is also $O(\log^2 n)$ space.

Algorithm 1: Checking connectivity of two vertices in an undirected graph

```
Input: An graph $G$, and vertices $s, t \in V(G)$
Output: Connectivity of $s, t$
1 isPath($s, t, k = 2^\ell$)
      // Outputs whether there is a path between $s$ and $t$ of length at most $k$
2     if $\ell = 0$ then
3         return yes iff $s, t$ are adjacent
4     foreach $v \in V(G)$ do
5         return yes iff isPath($s, v, 2^{\ell-1}$) and isPath($s, v, 2^{\ell-1}$)
6 return isPath($s, t, n$)
```

Observe that the depth of this recursion tree is $O(\log n)$. In each recursion call, we use $O(\log n)$ space. As a result, we use $O(\log^2 n)$ space in all.

All the algorithms we have presented thus far work in the setting of directed graphs as well.
Can we check connectivity in $O(\log n)$ space? We present a randomized algorithm due to Reingold [Rei08] that does so. The algorithm is as follows.

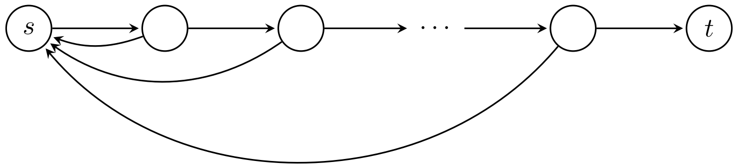
Algorithm 2: Checking connectivity of two vertices in an undirected graph

```
Input: An graph $G$, and vertices $s, t \in V(G)$
Output: Connectivity of $s, t$
1 $v \leftarrow s$
2 Uniformly randomly choose a neighbour $u$ of $v$
3 if $u = t$ then
4     return yes
5 else
6     $v \leftarrow u$
7     go to line 2
```

Suppose that G is connected. For the sake of simplicity, suppose that the graph is d-regular. If it is not, add self-loops to make it so. Also, after doing this, add another self-loop at each vertex.
We claim that

$$\Pr\Big[t \text{ is seen in } O(n^3 \log n) \text{ steps}\Big] \geq \frac{1}{2}.$$

This algorithm does *not* work for directed graphs. Indeed, consider the graph



where it takes exponential time for the probability of seeing t to go over $1/2$.

The transition matrix of the random walk is defined by

$$M_{ij} = \frac{1}{d} e(i,j),$$

where $e(i,j)$ is the number of edges between i and j.
Given the initial probability vector $x^{(0)} = \mathbb{1}_s$ ($x_s^{(0)} = 1$ and $x_v^{(0)} = 0$ for $v \neq s$), the probability distribution of vertices after t steps of the random walk is given by $x^{(t)} = M^t x^{(0)}$.
Consider the uniform probability vector u, where $u_v = 1/n$ for all v, and observe that $Mu = u$. u is called a stationary distribution of the random walk.
We claim that u is the only stationary distribution of the walk (this assumes that G is connected – why?).

2.2.2. Lecture 8

Denote by $p^{(t)}$ the probability vector at time t, and $p_i^{(t)}$ the probability that we are at vertex i at times t. By definition,

$$p_i^{(t)} = \frac{1}{d} \sum_{j \leftrightarrow i} p_j^{(t-1)}.$$

Now, we wish to analyze M^t to show that $p^{(t)}$ is close to the stationary distribution for somewhat large t.
Because M is symmetric, we can write $M = UDU^\top$, where U is an orthogonal matrix of eigenvectors and D is a diagonal matrix of the real eigenvalues. As a result,

$$p^{(t+1)} = (UDU^\top)^t p^{(0)} = UD^t U^\top p^{(0)}.$$

Therefore, what matters is D^t. Let the eigenvalues of M be $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.
We trivially have that $\lambda_1 = 1$ is an eigenvalue of M with eigenvector being the stationary distribution u_1 that we saw last lecture. Observe that 1 is the largest eigenvalue (in absolute value) of M because for any vector x, $\lambda \max_i x_i =$

$\max_i (Mx)_i \leq \max_i x_i$.
If $|\lambda_j| < 1$ for $j > 1$, then $\lim_{t\to\infty} p^{(t)} = u_1$ (all the other entries of D^t decay to 0).
To show fast convergence, we would like to show that the other eigenvalues are bounded away from 1.

Definition 2.8. Given a graph G, $\lambda(G) = \max\{|\lambda_2(G)|, |\lambda_n(G)|\}$. Equivalently,

$$\lambda(G) = \max_{x : \langle x, u_1\rangle = 0} \frac{\|Mx\|}{\|x\|}.$$

If G is obvious, we denote this as merely λ.

$(1-\lambda)$ is referred to as the *spectral gap* of the graph. We would like to show that the spectral gap of the graph is "large".
We have

$$\left\|p^{(t)} - u_1\right\| = \left\|M^t(p^{(0)} - u_1)\right\|$$

Observe that because $p^{(0)}$ is a probability distribution, $\langle p^{(0)}, u_1\rangle = 1/n = \langle u_1, u_1\rangle$, so $p^{(0)} - u_1$ is orthogonal to u_1. As a result,

$$\left\|p^{(t)} - u_1\right\| \leq \lambda^t \left\|p^{(0)} - u_1\right\|.$$

Suppose we want $\left\|p^{(t)} - u_1\right\| \leq 1/n^2$. For our specific choice of $p^{(0)}$ concentrated on a single point, it is easily verified that $\left\|p^{(0)} - u_1\right\| \leq 1$. Therefore, for $t > \log_\lambda \epsilon$, $\left\|p^{(t)} - u_1\right\| \leq 1/n^2$.
We have that the convergence time is

$$\log_\lambda \epsilon = O\left(\frac{1}{1-\lambda}\log n\right).$$

We would like to bound the spectral gap from below, perhaps by an inverse polynomial.

It is true that

$$\lambda_2 = \max_{\substack{x \perp u_1 \\ \|x\| = 1}} x^\top M x$$

Indeed, if $x = \alpha_2 u_2 + \cdots + \alpha_n u_n$ with $\sum \alpha_i^2 = 1$,

$$x^\top M x = \lambda_2 \alpha_2^2 + \cdots + \lambda_n \alpha_n^2 \leq \lambda_2.$$

We shall bound λ_2 for every graph, and use it for $H = G^2$, the multigraph with an edge from i to j if there was a length 2 walk from i to j. The random walk matrix on H is just M^2, so $\lambda_2(H) = \lambda(G)^2$. Now,

$$\begin{aligned}
x^\top M x &= \sum_i x_i \sum_{j \in \Gamma(i)} \frac{1}{d} x_j \\
&= \sum_{(i,i)\in E} \frac{x_i^2}{d} + \sum_{\substack{(i,j)\in E \\ i \neq j}} \frac{2x_i x_j}{d} \\
&= \sum_{(i,i)\in E} \frac{x_i^2}{d} + \sum_{\substack{(i,j)\in E \\ i \neq j}} \frac{x_i^2 + x_j^2 - (x_i - x_j)^2}{d} \\
&= \underbrace{\sum_{(i,j)\in E} \frac{x_i^2}{d}}_{1} + \sum_{\substack{(i,j)\in E \\ i \neq j}} \frac{(x_i - x_j)^2}{d}.
\end{aligned}$$

We want to bound the second expression from below. Because $\langle x, u_1 \rangle = 0$, there exist coordinates of both positive and negative sign. Further, by the pigeonhole principle, there exists some coordinate i_1 such that x_{i_1} is of absolute value at least $1/\sqrt{n}$. Assume that the sign of this coordinate is positive. Let i_k be a coordinate of negative sign, and $i_1 i_2 \cdots i_k$ a path from i_1 to i_k. Then, using the Cauchy-Schwarz inequality,

$$\begin{aligned}
\sum_{\substack{(i,j)\in E \\ i\neq j}} (x_i - x_j)^2 &= \sum_{1\leq j\leq (n-1)} (x_{i_{j+1}} - x_{i_j})^2 \\
&\geq \frac{1}{n}\left(\sum_{1\leq j\leq (n-1)} |x_{i_{j+1}} - x_{i_j}|\right)^2 \\
&\geq \frac{1}{n}\left(\sum_{1\leq j\leq (n-1)} x_{i_{j+1}} - x_{i_j}\right)^2 \geq \frac{1}{n^2}.
\end{aligned}$$

Therefore, the largest eigenvalue of any graph is at most $1 - 1/nd^2$.

2.2.3. Lecture 10

By arguments discussed in Lecture 8, we know that the random walk on an expander graph mixes in $O(\log n)$ time. This leads to a *deterministic* polynomial time algorithm to determine s, t connectivity in log-space – merely iterate over all paths of length $\log n$. That is, we iterate over all elements of $[d]^{\log n}$ and for each such element, we follow the corresponding path and check if we see t anywhere.

How do we extend this to arbitrary graphs? Given a graph G, we would like to get a graph G' such that

- G' has polynomially many vertices,
- G' has "constant" degree (independent of n),
- G' preserves connectivity (s, t are connected in G iff some s', t' are connected in G'), and
- each component of G' is an expander.

Let us look at a couple of operations one can do on a (n, d, λ)-expander G.

Squaring (G^2) the vertex set is the same as that of G, and there is an edge between u, v for each length 2 walk between $u, v \in G$. This is a multigraph with self-loops. G^2 is a (n, d^2, λ^2)-expander; the random walk matrix on G^2 is just M^2, where M is the random walk matrix on G. Because the second eigenvalue has gone down, connectivity has improved. However, the degree has increased.

Zig-zag product $(G \circledz H)$ Let G be a (n, D, λ_1)-expander and H a (D, d, λ_2)-expander. Note that the degree of the first graph is the number of vertices in the second! The goal of this product is to decrease the degree, without changing expansion too much.
The idea is very similar to that we saw in Lecture 9 to derandomize algorithms, and we shall "derandomize the random walk in G using H" – instead of going to a random neighbour in G, we determine which vertex to travel to using H.
$G \circledz H$ has nD vertices and is d^2-regular. Suppose that the edges at each vertex in G are put in bijection with $V(H)$ in some arbitrary manner.

- The vertex set of $G \circledz H$ is $V(G) \times V(H)$; we shall replace each vertex of G with the vertices of H.
- Let $v = (a_G, a_H) \in V(G \circledz H)$ and $(i, j) \in [d]^2$. The (i, j)th neighbour (b_G, b_H) of (a_G, a_H) is as follows.
  1. Let a'_H be the ith neighbour of a_H in H.
  2. Let (b_G, b'_H) be the a'_Hth neighbour of a_H in G (recall that we had put the neighbours of a given vertex in bijection with $V(H)$). That is, b_G is the a'_Hth neighbour of a_G and a_G is the b'_Hth neighbour of b_G.

3. Let b_H be the jth neighbour of b'_H.

Lemma 2.9. Suppose that H is a non-bipartite graph, and let w_H, w'_H be any vertices in H. Then, s, t are connected in G iff (s, w_H) and (t, w'_H) are connected in $G ⓩ H$.

Proof. It suffices to show that for any $v \in V(G)$, the "cloud" $v \times V(H)$ is connected. Indeed, the connectivity of clouds is identical to the connectivity of G. Consider some arbitrary a_H, b_H with a 2-walk $a_H v_H b_H$ in H between them. Due to the non-bipartiteness of H, we are done if we show that there is a path between (v, a_H) and (v, b_H). Indeed, considering some neighbour (w, c_H) of (v, a_H) in $G ⓩ H$ such that w is the v_Hth neighbour of v, it is seen that both (v, a_H) and (v, b_H) are adjacent to (w, c_H) in $G ⓩ H$, so we are done. ■

What happens to the eigenvalue of $G ⓩ H$? For the sake of simplicity, let $\gamma_1 = 1 - \lambda_1$ and $\gamma_2 = 1 - \lambda_2$ be the corresponding spectral gaps. We claim that

Lemma 2.10. It is true that
$$\gamma(G ⓩ H) \geq \gamma_1 \gamma_2^2.$$

We prove the above shortly, and first describe how this results in a conversion of our graph to an expander graph.

First of all, how do we construct the (D, d, λ_2) graph H?
There exists a simple method to construct a $(D^4, D, 1/8)$ graph H that we do not describe.
To generate a larger graph with bounded spectral value, we can do the following.

1. Set G_1 as H^2.
2. For each k, set G_{k+1} as $G_k^2 ⓩ H$.

We claim that G_k is a $(D^{4k}, D^2, 1/2)$ graph (by $1/2$ we mean that the eigenvalue is at most $1/2$).
The first two parameters are easily verified. It may be shown using Lemma 2.10 without much trouble that $\lambda(G ⓩ H) \leq \lambda_1 + 2\lambda_2$.
Then, an inductive argument yields the eigenvalue bound for G_k. Indeed, $\lambda(G_1) \leq 1/2$ and $\lambda(G_{k+1}) \leq \lambda(G_k)^2 + 2\lambda(H) \leq (1/2)^2 + 2/8 = 1/2$. Before coming to the proof of Lemma 2.10, we describe the construction used to convert G to a graph G' satisfying the conditions described earlier.

Algorithm 3: Converting an arbitrary graph to an expander

```
Input: An graph $G$
Output: A graph $G'$ each of whose components is an expander
1 $G_0 \leftarrow$ regularize$(G)$
2 for $1 \leq k \leq O(\log n)$ do
3     $G_{k+1} \leftarrow G_k^2 ⓩ H$
```

First, make G a regular graph. G_0 is an $(n, D^2, 1 - \frac{1}{\text{poly}(n)})$ graph. We claim that G_k is an $(nD^{4k}, D^2, 17/18)$.
The first two parameters are straightforward. Let $\gamma_k = 1 - \lambda(G_k)$. Assume that for some k $\gamma_{k-1} \leq 1/18$. Then,

$$\begin{aligned}\gamma_{k+1} &\geq \left(1 - (1 - \gamma_k)^2\right)\left(\frac{7}{8}\right)^2 \\ &= (2\gamma_k - \gamma_k^2)\frac{49}{64} \\ &\geq \left(\frac{35}{18}\gamma_k\right)\frac{49}{64} \geq \frac{5}{4}\gamma_k.\end{aligned}$$

Consequently, γ increases from $1/\operatorname{poly}(n)$ to a constant in $O(\log n)$ steps!
The only thing that remains is to prove Lemma 2.10.

Lemma 2.11. Let C be a random walk matrix with eigenvalue λ. Then, it is possible to write $C = (1-\lambda)J + \lambda E$ for some matrix E with spectral norm equal to 1.

Proof. Let v be a vector, and let v_1, v_2 be its components along and orthogonal to the all ones vector. Observe that all eigenvalues of J other than the first are equal to 0. As a result,

$$\begin{aligned}\|Ev\| &= \frac{1}{\lambda}\left\|Cv - (1-\lambda)Jv\right\| \\ &= \frac{1}{\lambda}\left\|v_1 + Cv_2 - (1-\lambda)v_1\right\| \\ &= \frac{1}{\lambda}\|\lambda v_1 + Cv_2\| \\ &= \frac{1}{\lambda}\sqrt{\lambda^2\|v_1\|^2 + \|Cv_2\|^2} \\ &\le \frac{1}{\lambda}\sqrt{\lambda^2\|v_1\|^2 + \lambda^2\|v_2\|^2} = \|v\|\,.\end{aligned}$$

The above inequality is tight on setting v_2 as the second eigenvector of C, so the spectral norm is precisely 1. ■

Proof of Lemma 2.10. Let B be the random walk matrix of H, and let $\tilde{B} = I_n \otimes B$.[2] Let $\tilde{A}$ be the matrix that encodes the second edge (using G-edges) traversed in the construction of the zigzag product. Then, the transition matrix M of $G \,ⓩ\, H$ is just $\tilde{B}\tilde{A}\tilde{B}$!
Now, denote by J the matrix that has all elements $1/n$.
Now, use Lemma 2.11 on $\tilde{B}$.

$$\begin{aligned}M &= \tilde{B}\tilde{A}\tilde{B} \\ &= \left(\gamma_2\tilde{J} + \lambda_2\tilde{E}\right)\tilde{A}\left(\gamma_2\tilde{J} + \lambda_2\tilde{E}\right) \\ &= \gamma_2^2\tilde{J}\tilde{A}\tilde{J} + \gamma_2(1-\gamma_2)\left(\tilde{E}\tilde{A}\tilde{J} + \tilde{J}\tilde{A}\tilde{E}\right) + (1-\gamma_2)^2\tilde{E}\tilde{A}\tilde{E} \\ &= \gamma_2^2\tilde{J}\tilde{A}\tilde{J} + (1-\gamma_2^2)X,\end{aligned}$$

for some matrix X. Observe that the spectral norm of X is at most 1 because $\tilde{E}, \tilde{A}, \tilde{J}$ all have spectral norms at most 1, and so by submultiplicativity of the spectral norm,[3] $\tilde{E}\tilde{A}\tilde{J}$, $\tilde{J}\tilde{A}\tilde{E}$, and $\tilde{E}\tilde{A}\tilde{E}$ all have spectral norms at most 1.
Now, we wish to bound the second eigenvalue of M. Note that $\tilde{J}\tilde{A}\tilde{J}$ is precisely the random walk matrix of $G \otimes K_n$, which has second eigenvalue equal to that of G. Recalling very carefully that the second eigenvector of M is some vector v orthogonal to $\mathbf{1}$, we have

$$\begin{aligned}\|Mv\|_2 &= \left\|\gamma_2^2\tilde{J}\tilde{A}\tilde{J}x\right\| + \left\|(1-\gamma_2^2)Xx\right\| \\ &\le \left(\gamma_2^2\lambda_1 + (1-\gamma_2^2)\right)\|v\| = (1-\gamma_2^2\gamma_1)\|v\|\,,\end{aligned}$$

completing the proof. ■

2.3. Lecture 9: Expander graphs

2.3.1. Lecture 9

Definition 2.12. A graph G is said to be an (n,d,λ)-*expander* if $|V(G)| = n$, G is d-regular, and λ is the second largest eigenvalue $\lambda(G)$ of G in absolute value.

[2] this is a $nD \times nD$ matrix, with n blocks on the diagonal that are all Bs.
[3] $\|AB\| \le \|A\|\|B\|$. This is obvious because $\|ABx\| \le \|A\|\|Bx\| \le \|A\|\|B\|\|x\|$.

Definition 2.13 (Spectral expanders)**.** A sequence $\{G_n\}_{n\geq 0}$ of d-regular graphs is said to be a *spectral expander family* if for some $\lambda < 1$, $\lambda(G_i) \leq \lambda$ for all i.

We saw last lecture that random walks on expander graphs converge to the uniform distribution in $O(\log n)$ steps. This means that there are only $d^{O(\log n)} = \operatorname{poly}(n)$ paths to explore. Therefore, there is a *deterministic* polynomial time algorithm to determine connectivity of expander graphs.

Definition 2.14 (Sparsity)**.** Given a d-regular graph G, define the *sparsity*

$$h(G) = \min_{\substack{S\subseteq V \\ |S|\leq (n/2)}} \frac{|E(S,\overline{S})|}{d|S|}.$$

Some definitions use $|E(S,\overline{S})|/|S|$ instead.
It is natural to see that $h(G)$ measures (in some sense) how well-connected a graph is. If it is low, there is some "bottleneck" in the graph where the random walk can get stuck – a set of high measure with very few outgoing edges.
It is clear that $h(G) \leq 1$.

Definition 2.15 (Combinatorial expanders)**.** A sequence $\{G_n\}_{n\geq 0}$ of d-regular graphs is said to be a *combinatorial expander family* if for some $h > 0$, $h(G_i) \geq h$ for all i.

Theorem 2.16 (Cheeger's Inequality)**.** For any graph G with second eigenvalue λ_2 and sparsity h,

$$\frac{1-\lambda_2}{2} \leq h \leq \sqrt{2(1-\lambda_2)}.$$

In particular, spectral expanders are combinatorial expanders and vice-versa.

We prove this later.

Markov chain Monte Carlo methods find many uses nowadays in problems such as sampling random spanning trees, random independent sets etc. The idea in these is that we start with an arbitrary spanning tree (say), and then randomly move to a "neighbouring" spanning tree – add a random edge not in the spanning tree and remove a random edge from the cycle thus formed. After sufficiently many steps, we are at a(n almost) uniformly random spanning tree. This massive graph composed of spanning trees as vertices ends up being an expander! Because the graph of spanning trees has only exponentially many vertices, we get a polynomial time algorithm to randomly sample spanning trees.

Example 2. The n-cycle C_n has sparsity $h(C_n) = 2/n$, and $\lambda(C_n) = \cos(2\pi/n) \approx 1 - (2\pi/n)^2$.
The hypercube graph $H_n := P_2^{\otimes n}$. We have $h(H_n) = 1/n$ and $\lambda(H_n) = 1 - \frac{1}{k}$.
Each of these give a case where the appropriately inequality in Cheeger's Inequality is (asymptotically) tight.

What guarantee do we even have that expanders exist? It turns out that a random d-regular graph is a (combinatorial) expander with high probability!

However, how do we construct expander graphs? Our goal is to use expander graphs to reduce randomness in algorithms, so it does not make sense to construct them using the above random argument. We also want the algorithm itself to run in polylog time – this requirement makes sense in light of our remarks towards the end of Lecture 6.

Example 3. Let p be a prime and consider the 3-regular graph over $\mathbb{F}_p^*$, where each x is adjacent to $x+1, x-1, x^{-1}$. This graph is an expander, but the proof of this is not very straightforward.

Theorem 2.17 (Expander Mixing Lemma)**.** Let G be a (n, d, λ)-expander. Then, for any $S, T \subseteq V$,

$$\left| E(S,T) - \frac{d}{n}|S||T| \right| \le d\lambda\sqrt{|S||T|}.$$

If G were a random graph, then the expected number of edges between S, T is precisely $(d/n)|S||T|$ – of the $d|S|$ edges out of S, we expect a $|T|/n$ fraction to be incident on T.

Proof. Let M be the transition matrix of the random walk on G; it is equal to $(1/d)$ times the adjacency matrix of G. For any set X, let $\mathbb{1}_X$ be the indicator vector of X with 1s at vertices in X and 0 elsewhere.
Observe that

$$\frac{1}{d}E(S,T) = \mathbb{1}_S^\top M \mathbb{1}_T.$$

Now, we have $M = \sum_i \lambda_i u_i u_i^\top$ using the spectral theorem, where (u_i) are orthonormal eigenvectors of M with corresponding real eigenvalues (λ_i). Note in particular that $\lambda_1 = 1$ and u_i is the vector with all coordinates having value $1/\sqrt{n}$.
Let $\mathbb{1}_S = \sum_i \alpha_i u_i$ and $\mathbb{1}_T = \sum_i \beta_i u_i$. Note in particular that $\alpha_1 = \langle \mathbb{1}_S, u_1 \rangle = |S|/\sqrt{n}$ and $\beta_1 = |T|/\sqrt{n}$.
Using orthonormality,

$$\begin{aligned}\frac{1}{d}E(S,T) &= \left(\sum_i \alpha_i u_i\right)\left(\sum_i \lambda_i u_i u_i^\top\right)\left(\sum_i \beta_i u_i\right) \\ &= \sum_i \alpha_i \beta_i \lambda_i \\ &= \alpha_1\beta_1 + \sum_{i=2}^n \alpha_i\beta_i\lambda_i \\ &= \frac{|S||T|}{n} + \sum_{i=2}^n \alpha_i\beta_i\lambda_i.\end{aligned}$$

Therefore,

$$\begin{aligned}\left|E(S,T) - \frac{d}{n}|S||T|\right| &= \left|\sum_{i=2}^{n} \alpha_i\beta_i\lambda_i\right| \\ &\leq d\lambda \sum_{i=2}^{n} |\alpha_i\beta_i| \\ &\leq d\lambda \sqrt{\left(\sum_{i=2}^{n} \alpha_i^2\right)\left(\sum_{i=2}^{n} \beta_i^2\right)} \\ &\leq d\lambda\sqrt{\|\mathbb{1}_S\|\|\mathbb{1}_T\|} = d\lambda\sqrt{|S||T|}.\end{aligned}$$

■

We now see how to save randomness using expanders.
Let $\mathcal{A}$ be an algorithm that uses k independent random bits. Let G be a $(2^k, d, \lambda)$-expander. Starting at an arbitrary vertex v_1, perform a random walk for ℓ steps through vertices $v_1, v_2, \ldots, v_\ell$. Run the algorithm on each of these inputs $v_1, \ldots, v_\ell$ (interpreting the 2^k elements of $V(G)$ as length k bit strings.
Recall that if $\mathcal{A}$ once (using k bits) has error probability β, running the algorithm ℓ times (using $k\ell$ bits) reduces this to error probability β^ℓ. It turns out that running the algorithm ℓ times as described above (using $k + \ell \log d$ bits) reduces the error probability to $(\beta + \lambda)^\ell$!
In purely graph theoretic terms, this says the following.

Theorem 2.18. Let G be a (n, d, λ)-expander, and let $B \subseteq V$ be of size βn. Starting at a random vertex v_1, consider ℓ steps of the random walk going through vertices $v_1, v_2, \ldots, v_\ell$. Then,

$$\Pr\left[\text{all } v_i \text{ are in } B\right] \leq (\beta + \lambda)^\ell.$$

Proof sketch. Consider the diagonal matrix D with 1s at vertices in B and 0 elsewhere. Let $p^{(0)}$ be the initial (uniform) distribution of v_0. Observe that the ℓ_1 norm of $(BM)^i p^{(0)}$ is precisely the probability that v_i is in B. To bound this, we split a given vector into its component along and orthogonal to the uniform distribution. The norm of the second part decreases by λ at every step. ■

Theorem 2.19 (Alon-Boppana bound)**.** For any (n, d, λ)-expander,

$$\lambda = \frac{2\sqrt{d-1}}{d}(1 - o(1)).$$

Definition 2.20 (Ramanujan Graph)**.** A (n, d, λ)-expander is said to be a *Ramanujan graph* if

$$\lambda \leq \frac{2\sqrt{d-1}}{d}.$$

It was proved in 2014 by Adam Marcus, Daniel Spielman, and Nikhil Srivastava that there exist infinite families of bipartite Ramanujan graphs of every degree greater than 2.

§3. Reed-Solomon Codes

3.1. Lecture 11: Introduction

3.1.1. Lecture 11

Randomly choosing strings from $\{0,1\}^n$ tends to yield a good code, with high distance between points with high probability. This is true even for exponentially many points, say $2^{0.1n}$. At the heart of getting good codes is derandomizing this process to get good explicit codes.
In some sense, good codes, expander graphs, and extractors are equivalent. In fact, some of the best known expander constructions today come from coding theoretic constructions.
The subject of coding theory lies at the intersection of numerous disparate fields, such as (theoretical) computer science, electrical engineering, and math. Interestingly, the same objects are studied in all the disciplines, merely from different perspectives.

Definition 3.1 (Reed-Solomon Code)**.** Let $\mathbb{F}$ be a finite field, and $k, n \in \mathbb{N}$ with $n \geq k, |\mathbb{F}|$. Also fix some distinct $\alpha_1, \ldots, \alpha_n$. The message space of the *Reed Solomon code* $\mathrm{RS}(k,n)$ is

$$\{g(x) \in \mathbb{F}[x] : \deg(g) \leq k-1\}.$$

That is, we identify k-dimensional vectors in $\mathbb{F}^k$ with corresponding polynomials. A polynomial g is encoded as

$$\mathrm{Enc}(g) = (g(\alpha_1), \ldots, g(\alpha_n)).$$

Note that the number of possible messages if $|\mathbb{F}|^k$. Let us look at some basic properties of this code.

1. A Reed-Solomon code is a linear code. This follows immediately from the fact that $(\alpha g+h)(\alpha_i) = \alpha g(\alpha_i)+h(\alpha_i)$, and if g, h are of degree at most $k-1$ then so is $\alpha g + h$.
2. The code has rate k/n.
3. The distance of the code is $n-k+1$. Indeed, by the Fundamental Theorem of Algebra, two polynomials can coincide in value at at most $k-1$ points (otherwise their difference, a nonzero polynomial of degree at most $k-1$, would have more than $k-1$ roots).

Observe that given a vector $(g_1, \ldots, g_k) \in \mathbb{F}^k$, we encode it as

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{k-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{k-1} \end{pmatrix} \begin{pmatrix} g_1 & g_2 & \vdots & g_k \end{pmatrix}.$$

Also note that the above properties do not depend on our choice of (α_i).

It turns out that Reed-Solomon codes are optimal in some sense.

Theorem 3.2. Reed-Solomon codes match the singleton bound.

This says that over large fields, they essentially match the best possible rate-distance trade-off.

3.2. Lecture 11-12: Decoding

3.2.1. Lecture 11 (continued)

The matrix multiplication scheme above describes a simple way to encode RS codes. How do we decode them? That is, if $r = (r_1, \dots, r_n) \in \mathbb{F}^n$, what do we decode it to? This asks to find the RS codeword c closest to r; find *the* RS codeword c such that $d_H(c, r) < (n-k+1)/2$ (if no such codeword exists, say no).

In the 60s, numerous decoding algorithms such as those of Peterson, Sugiyama, and Berlekamp-Massey were proposed. Despite being elementary, these are all very clever. In practice, Berlekamp-Massey is typically used (it is nearly linear in the input size).
We shall look at the Welch-Berlekamp algorithm, which was proposed in the 80s. Amusingly, this was not originally published in a paper but in a patent. Some time later, Madhu Sudan et al. decoded[4] the patent. The algorithm has had surprisingly pervasive effects in computer science, and the methods have helped resolve several open problems in math as well. Interested readers can see the "polynomial method" for more details; while it has been used in the past in math, this introduced it to mainstream theoretical computer science.
The algorithm is as follows.

1. Find a nonzero polynomial $Q(x, y) = A(x) + yB(x)$ such that
   (a) $\deg(B) < \lceil (n-k+1)/2 \rceil =: d$,
   (b) $\deg(A) < d + (k-1) =: D$, and
   (c) for all i, $Q(\alpha_i, r_i) = 0$.
2. Set $g(x) = -A(x)/B(x)$. If g is a polynomial and $d_H(\text{Enc}(g), r) < (n-k)/2$ and $\deg(g) \leq k-1$, output g. Otherwise, say that no codeword exists.

Observe that if g is a polynomial, the polynomial is certainly correct.
We must prove that if g is not a polynomial, then no codeword exists. First, however, let us describe how to determine the bivariate polynomial Q.
Set $D = (n+k)/2$ and $d = (n-k)/2$. The desired Q is of the form

$$Q(x, y) = A_0 + A_1 x + \cdots + A_{D-1} x^{D-1} + B_0 y + B_1 xy + \cdots + B_{d-1} x^{d-1}.$$

Then, we wish to determine coefficients (A_i) and (B_i) such that for all i

$$Q(x, y) = A_0 + A_1 \alpha_i + \cdots + A_{D-1} \alpha_i^{D-1} + B_0 r_i + B_1 r_i \alpha_i + \cdots + B_{d-1} r_i \alpha_i^{n-1} = 0.$$

This is just a linear system of n equations. If $D + d > n$, which is indeed true, we may solve it for a non-trivial solution.
To compute g using A, B, we just have to do polynomial long division from high school (there are better methods to do this).
All that remains is to check correctness.

Theorem 3.3. If there exists a polynomial $h \in \mathbb{F}[x]$ of degree at most $k-1$ such that $d_H(\text{Enc}(h), r) < (n-k+1)/2$, the Welch-Berlekamp algorithms outputs it.

Proof. Let $Q(x, y) = A(x) + yB(x)$ with $\deg(A) < D$, $\deg(B) < d$, and $Q(\alpha_i, r_i) = 0$ for all i be the polynomial output in the first step of the algorithm.

Consider $U(x) = Q(x, h(x))$. This is $A(x) + h(x)B(x)$. For starters, the degree of U is at most $D - 1 < (n+k-1)/2$. If $h(\alpha_i) = r_i$, then $U(\alpha_i) = 0$. Consequently, the number of zeroes of U on $\{\alpha_1, \dots, \alpha_n\}$ is at least the number of agreements between $\text{Enc}(h)$ and r. Due to the distance contraint on h, there are at least $(n+k-1)/2$ such agreements. Therefore, U has more zeros than degree, so U must be the zero polynomial (!) and therefore g is indeed a polynomial and equal to h. ■

[4]Pardon the pun.

The idea of the construction is just that the first step forces us to have a Q of large degree, while the second (assuming a valid h exists) forces Q to have small degree. The sweet spot in the middle is precisely where we lie.

Next class, we shall look at decoding Reed-Solomon codes beyond the error limit of half the minimum distance. One option is to output any codeword in the given radius. The more useful (albeit more stringent) notion is to output all these codewords. Such algorithms are called "list decoding algorithms". We must ensure that the amount we go beyond $(n-k+1)/2$ is not so high that the list becomes exponentially large. This is guaranteed by the following.

Theorem 3.4 (Johnson)**.** Let $\mathcal{C}$ be a code of block length n with distance Δ. Then, for all $r \in \mathbb{F}^n$, the number of codewords in $\mathcal{C}$ within distance $n - \sqrt{n(n-\Delta)}$ (the "Johnson radius") of r is $\operatorname{poly}(q, n, \Delta)$.

For Reed-Solomon codes, this value is equal to

$$n - \sqrt{n\left(n - (n-k+1)\right)} \approx n - \sqrt{nk}.$$

This is *far* larger than the minimum distance. If $k = 0.01n$, say, then the minimum distance is about $0.49n$, but the Johnson radius is about $0.9n$! Next class, we shall describe how to output all the codewords within the prescribed radius.
We also remark that there exist Reed-Solomon codes where we go beyond the Johnson radius while still having polynomially many codewords in the given radius. Such codes are not *explicitly* known, however, and are obtained by taking a sufficiently large field and choosing random evaluatin points α_i. The question of when in general the Johnson bound is not tight does not have many satisfactory answers to date. In particular, we do not know if the Johnson bound is non-tight for *all* Reed-Solomon codes.

3.2.2. Lecture 12

Given $\mathcal{S} = \{(\alpha_i, \beta_i)\}_{i=1}^n$, we aim to find *all* polynomials $f \in \mathbb{F}_q[x]$, $\deg f < k$, such that

$$\operatorname{agr}(f, \mathcal{S}) := \left|\left\{u \in [n] : f(\alpha_i) = \beta_i\right\}\right| \geq t.$$

Last lecture, we saw the setting where $t \geq (n+k-1)/2$.
Due to Johnson, the setting where $t \geq \sqrt{nk}$ enters consideration. We shall now look at how to decode Reed-Solomon codes up to the Johnson radius.

First, we shall look at the case where $t \geq 2\sqrt{nk}$. The basic idea is that we take the Welsh-Berlekamp algorithm, but look at polynomials that are higher degree in y.

1. Set $\ell \approx \sqrt{n/(k-1)}$. Find nonzero $Q(x,y) = A_0(x) + A_1(x) + y^2 A_2(x) + \cdots y^\ell A_\ell(x)$ such that $\deg(A_i) \leq n/\ell$ for each i, and $Q(\alpha_i, \beta_i) = 0$ for each i.
2. Find all factors of $Q(x,y)$ of the form $(y - h(x))$, where $\deg(h) < k$ and $\operatorname{agr}(h, \mathcal{S}) \geq t$. Output all such h.

This is very similar in spirit to the earlier algorithm. Indeed, $Q(x, f(x)) = 0$ just says that $(y - f(x))$ divides $Q(x,y)$. The first step of the algorithm is exactly as earlier and amounts to solving a system of linear equations. There is an algorithm, that factors polynomials of $\deg d$ on $\mathbb{F}_q$ in time $\operatorname{poly}(d, \log q)$.[5] This is rather interesting, as it means we are able to factorize elements of the polynomial ring. Compare this to the integer ring, where we cannot factorize elements efficiently. We do not describe this algorithm.

Proof of correctness. The number of variables in the system of linear equations in the first step is $(\ell+1)\left(\frac{n}{\ell}+1\right) > n$, which is more than the number of datapoints, so such a Q exists.
Let $f \in \mathbb{F}[x]$ be of degree $< k$ and agree with $\mathcal{S}$ at more than $2\sqrt{nk}$ points. Then, we wish to show that $R(x) :=$

[5]This algorithm is randomized, and no such deterministic algorithm is known

$Q(x, f(x)) \equiv 0$. We have $\deg(R) \leq (k-1)\ell + n\ell$. On the other hand, as before, if $f(\alpha_i) = \beta_i$, $R(\alpha_i) = Q(\alpha_i, \beta_i) = 0$. If the number of agreements is more than $\deg(R)$, we are done. That is,

$$t > \frac{n}{\ell} + (k-1)\ell.$$

The quantity on the right is minimized for $2\sqrt{n(k-1)}$ and the corresponding ℓ is approximately $\sqrt{n/(k-1)}$. ■

Now, let us modify the algorithm slightly to $\sqrt{2nk}$.

1. Set $D \approx \sqrt{2n(k-1)}$. Find nonzero $Q(x,y) = A_0(x) + A_1(x) + y^2 A_2(x) + \cdots y^{D/(k-1)} A_{D/(k-1)}(x)$ such that $\deg(A_i) \leq D - (k-1)i$ for each i, and $Q(\alpha_i, \beta_i) = 0$ for each i.
2. Find all factors of $Q(x,y)$ of the form $(y - h(x))$, where $\deg(h) < k$ and $\operatorname{agr}(h, \mathcal{S}) \geq t$. Output all such h.

Proof of correctness. The key observation is that in the above argument, we can push the degree of most A_i higher, without affecting the bound on the overall degree. Let $\deg A_i = D_i$, so the degree of $f^i A_i$ is at most $i(k-1) + D_i$. If we want the overall degree to be D, then we get $\deg(A_i) \leq D - (k-1)i$.
The new number of variables is approximately

$$\sum_{i=0}^{D/(k-1)} D - (k-1)i \approx \frac{D^2}{k-1} - \frac{D^2}{2(k-1)} = \frac{D^2}{2(k-1)}.$$

So, we want a D such that the above is greater than D.
For the second part of the argument, we have $\deg(R) \leq D$ by definition, so we are fine if $t > D$.
Overall, this gives a bound of around $\sqrt{2n(k-1)}$.

■

Finally, let us look at how to get a bound of $t \geq \sqrt{nk}$. This argument is slightly more involved than the short jump it took to get from $2\sqrt{nk}$ to $\sqrt{2nk}$.
We shall begin with a brief discussion of the *method of multiplicities*, which is something like the polynomial method on steroids.

Definition 3.5. A polynomial $Q(x,y)$ is said to have a zero of multiplicity $\geq r$ at (α, β) if for all i, j such that $i + j < r$,

$$\frac{\partial Q}{\partial x^i y^j}(\alpha, \beta) = 0.$$

1. Set D, r such that $D/r \approx \sqrt{n(k-1)}$. Find nonzero $Q(x,y) = A_0(x) + A_1(x) + y^2 A_2(x) + \cdots y^{D/(k-1)} A_{D/(k-1)}(x)$ such that $\deg(A_i) \leq D - (k-1)i$ for each i, and Q passes through (α_i, β_i) with multiplicity r.
2. Find all factors of $Q(x,y)$ of the form $(y - h(x))$, where $\deg(h) < k$ and $\operatorname{agr}(h, \mathcal{S}) \geq t$. Output all such h.

Proof of Correctness. The number of variables now remains $\frac{D^2}{2(k-1)}$, but the number of constraints has increased to about $\binom{r+1}{2}n$. So, we require

$$\frac{D^2}{2(k-1)} \geq \binom{r+1}{2} n.$$

Approximately, this requires.

$$\frac{D^2}{r^2} \geq n(k-1).$$

Now, due to our additional multiplicity constraints, if $f(\alpha_i) = \beta_i$, then $R(x) = Q(x, f(x))$ vanishes with multiplicity at least r at α_i. Now, we have that there are at most D/r such distinct f. The observation is that each point of

agreement gives us r zeros, not just one.
We require

$$t \geq \frac{D}{r}.$$

In all, this gives us the required bound $\sqrt{n(k-1)}$! ■

We again draw attention to the part where we used the fact that a nonzero degree d univariate polynomials has at most d/r distinct zeros of multiplicity $\geq r$. Using this fact, we can consider another code.

3.3. Lecture 14: Multiplicity codes

3.3.1. Lecture 12 (continued)

Definition 3.6 (Univariate Multiplicity code)**.** Let $\mathbb{F}$ be a finite field of size at least n, $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. The message set is $\{f \in \mathbb{F}[x], \deg f < k\}$. We map f to the n-dimensional vector M over $\mathbb{F}^s$, where

$$(M_i)_j = f^{(j)}(\alpha_i) = \frac{\partial^{j-1} f}{\partial x^{j-1}}(\alpha_i).$$

Note that the messages are encoded in $\mathbb{F}^s$, so errors mean errors in the entire vector, not specific derivatives.
The rate of this code is approximately k/ns, which is worse than before. The distance however, jumps up to $n - \frac{k-1}{s}$!
A unique decoding algorithm for the multiplicity is very similar to Berlekamp-Welch, and we omit the details.
Next class, we shall prove incredible list decoding results, namely that for any $\epsilon > 0$, for sufficiently large s, multiplicity codes can be efficiently decoded from fractional agreement $\frac{k}{ns} + \epsilon$. We can get arbitrarily close to the (hard) bound – we cannot hope to get a degree k polynomial with fewer than k datapoints!
Recent state-of-the-art expander graphs are constructed using multiplicity codes!

3.3.2. Lecture 14

When talking about the derivative, we mean the *syntactic* derivative, which evaluates (on exponents of x) exactly the same as ordinary derivatives.
Note that in contrast to Reed-Solomon codes, we can allow the degree of the polynomial to be more than n.

Theorem 3.7 (Neilsen '01, Kopparty '13, Guruswami-Wang 14)**.** For every $\epsilon > 0$, for sufficiently large s, univariate multiplicity codes are efficiently list decodable from fractional agreement $\frac{k}{ns} + \epsilon$.

We can get arbitrarily close to the (hard) bound (!) – we cannot hope to get a degree k polynomial with fewer than k datapoints. Further, this can be done with a constant list size, with the constant depending on ϵ – this was shown by Kopparty-Saraf-RonZewi-Wooffer '17.
The fraction of agreement here is $(1+\epsilon)\frac{k}{sn} = (1+\epsilon) \cdot (\text{Rate})$. Compare this to what we had studied about Reed-Solomon codes, where we only had $\sqrt{\text{Rate}}$.

The remainder of this section is dedicated to the proof of this theorem; we shall look at the proof/algorithm of this due to Guruswami-Wang.

The input to the algorithm is an $s \times n$ matrix Y. We wish to find all polynomials p of degree at most k whose encoding has "large" agreement with Y. More precisely, there is a set $T \subseteq [n]$ of size greater than t such that for all $i \in T$ and $j \in [s]$,

$$p^{(j)}(\alpha_i) = Y_{ji}.$$

Call this set of all polynomials as $\mathcal{L}$. We want t to be as small as possible.
Sticking with the Welch-Berlekamp idea, the proof (and algorithm) goes as follows.

1. Find a nonzero $(m+2)$-variate polynomial
$$Q(x, z_0, z_1, \dots, z_m) = z_0 A_0(x) + z_1 A_1(x) + \dots + z_m A_m(x)$$
such that
   - $\deg(A_i) < D$ for some D we shall fix later,
   - multiplicity constraints which we shall come up with later, and
   - Q "explains" the given data: for every $j \in [n]$, $Q(\alpha_i, Y_{0,i}, Y_{0,i}, \dots, Y_{m,i}) = 0$. We want it to explain the top m rows of the matrix,
2. Show that for all $p \in \mathcal{L}$,
$$Q(x, p(x), p^{(1)}(x), \dots, p^{(m)}(x)) \equiv 0. \tag{3.1}$$
3. Find all low degree solutions to Q satisfying Equation (3.1). Note that we cannot rely on factoring for this, and it is more complicated.

Set $R(x)$ equal to the LHS of Equation (3.1) for some polynomial p, so it is
$$R(x) = A_0 p + A_1 p^{(1)} + \dots + A_m p^{(m)}.$$
If Y and the encoding of p agree at α_i, then $R(\alpha_i) = 0$.[6] The multiplicity constraint means that we also want the derivative of R to be zero at α_i. We have
$$\frac{\mathrm{d}R}{\mathrm{d}x} = A_0^{(1)} p + A_0 p^{(1)} + A_1^{(1)} p^{(1)} + A_1 p^{(2)} + \dots + A_m^{(1)} p^{(m)} + A_m p^{(m+1)},$$
so if $m < s$,
$$0 = \left.\frac{\mathrm{d}R}{\mathrm{d}x}\right|_{\alpha_i} = A_0^{(1)}(\alpha_i) Y_{0,i} + A_0(\alpha_i) Y_{1,i} + A_1^{(1)}(\alpha_i) Y_{1,i} + A_1(\alpha_i) Y_{2,i} + \dots + A_m^{(1)}(\alpha_i) Y_{m,i} + A_m(\alpha_i) Y_{(m+1),i}.$$
So, at each i, the aforementioned multiplicity constraints correspond to about $s - m - 1$ additional constraints of the above form.

Now, we would like to set D in the first step such that it has a solution. There are Dm variables and $n(s-m-1)$ constraints. So, we require $Dm \geq n(s-m-1)$. Set
$$D = \frac{n}{m}(s-m).$$
Let us now look at step 2. For a given polynomial in $\mathcal{L}$, the degree of R is at most $D + k - 1$. To ensure that R is identically zero, we need that $t(s-m-1) \geq D + k$, since each point of agreement gives $(s-m-1)$ equations. That is, we need
$$\begin{aligned} t &> \frac{1}{s-m}(D+k) \\ &\approx \frac{n}{m} + \frac{k}{s-m} \\ \frac{t}{n} &> \frac{k}{n(s-m)} + \frac{1}{m}. \end{aligned}$$
Setting m as around $1/\epsilon$ and $s > 1/\epsilon^2$ does the job!

[6]Stopping here would lead to unique decoding, by setting m as s or $s-1$ or so.

Finally, it remains to see if it is possible to find all low degree solutions p to $Q(x, p, p^{(1)}, \dots, p^{(m)}(x)) \equiv 0$. Let us look at just the trivariate case, with $Q(x, p, p') \equiv 0$. That is, we wish to solve

$$A_0(x)p(x) + A_1(x)p^{(1)}(x) + A_2(x)p^{(2)}(x) \equiv 0.$$

Note that the space of all p satisfying this is a subspace of the space of all polynomials of degree $< k$. We may assume wlog that two of the A_i are nonzero, as the problem is not very interesting otherwise. Suppose that $A_2 \not\equiv 0$. This means that there exists some $\beta \in \mathbb{F}$ such that $A_2(\beta) \neq 0$. We can assume wlog that $\beta = 0$ by "shifting" the axis by β otherwise. Dividing by a constant, we can also assume that the constant term in A_2 is 1, so

$$A_0 p + A_1 p^{(1)} + (1 + \tilde{A}_2)p^{(2)} \equiv 0,$$

where $\tilde{A}_2$ has no constant term.
The p we wish to find is of the form

$$p(x) = p_0 + p_1 x + p_2 x^2 + \cdots p_{k-1} x^{k-1}.$$

Plugging this into the previous equation, we have

$$A_0(p_0 + p_1 + \cdots) + A_1(p_1 + 2p_2 x + \cdots) + (1 + \tilde{A}_2)(2p_2 + 3 \cdot 2p_3 + \cdots) \equiv 0.$$

This means that all the coefficients of the resulting polynomial is zero. This is just a linear system of equations, so we can solve it. It remains to argue that the number of solutions (the list size) is not too large.
The coefficient for the degree 0 coefficient is

$$A_{00}p_0 + A_{10}p_1 + 2p_2 = 0,$$

In fact, note that the equation for the coefficient of degree k being 0 involves only the first $k + 2$ coefficients of p! Consequently, the solution space lives in a 3-dimensional subspace, so it is solvable. In general, it lives in an $(m+1)$-dimensional subspace.
These constructions give the best bipartite expanders, condensers, and extractors that we know.

§4. Hardness v. Randomness

4.1. Lecture 13, 15: Hardness

4.1.1. Lecture 13

A general question one can ask is this: for any polynomial-time randomized algorithm, is it possible to derandomize it (possibly using more space) to get a polynomial-time deterministic algorithm doing the same job?
It has been shown that given a "hard" function, one can construct very good pseudorandom bits. However, no explicit hard functions are known.

Consider the notion of *worst case hardness*. For example, if we can show for some language that no algorithm that runs in $O(n^{10})$ can compute the output correctly on all inputs, then the language is hard in some sense.
We also have the notion of *average case hardness*. Here, if we can show for some language that no algorithm that runs in $O(n^{10})$ can compute the output correctly on more than $3/4$ of the inputs, then the language is hard in some sense.
It is not too difficult to see that average case hardness is a stronger notion than worst case hardness.

Problems that are average case hard yield good pseudorandom generators. Another question of concern is converting worst case hardness to average case hardness, which is done through error correcting codes.
The rough idea behind the second question is the following. Error-correcting codes, when given two words as input that are close, make them far apart.

Definition 4.1 (RP). A language L is said to be in RP if there is a randomized algorithm $\mathcal{A}$ running in polynomial time such that

1. for $x \in L$,
$$\Pr_r\left[\mathcal{A}(x,r) = \text{yes}\right] \geq \frac{1}{2}.$$
2. for $x \notin L$,
$$\Pr_r\left[\mathcal{A}(x,r) = \text{yes}\right] \geq 0.$$

For example, the algorithm we saw in Section 2.2 was in RP.

Definition 4.2 (BPP). A language L is said to be in BPP if there is a randomized algorithm $\mathcal{A}$ running in polynomial time such that

1. for $x \in L$,
$$\Pr_r\left[\mathcal{A}(x,r) = \text{yes}\right] \geq \frac{2}{3}.$$
2. for $x \notin L$,
$$\Pr_r\left[\mathcal{A}(x,r) = \text{yes}\right] \leq \frac{1}{3}.$$

Here, by randomized algorithms, we mean probabilistic turing machines.

Next, let us look at pseudorandom distributions. The goal of these is to find some universal set of random bits which we can substitute in place of the (ideally) uniformly random bits.
Denote by U_m the uniform distribution on $\{0,1\}^m$.

The idea of this is that a distribution D is pseudorandom (with respect to $\mathcal{A}$) if for the given algorithm $\mathcal{A}$, for all inputs x,

$$\left| \Pr_{r \sim U_m}[B(x,r) = \mathsf{yes}] - \Pr_{r \sim D}[B(x,r) = \mathsf{yes}] \right| \le \epsilon.$$

One neat way to think about algorithms *with input* is circuits.
In fact, any deterministic algorithm can be viewed as a family $(C_n)_{n \ge 1}$ of circuits, with C_n computing the output correctly if the input is of size n. We can view a randomized algorithm as a deterministic one with two inputs, x and r. The circuit then has $n+m$ input leaves, where n is the size of the input x and m is the number of random bits r. If we fix the x part of the input, we get a circuit in the remaining input, namely $r_1, \dots, r_m$.

Definition 4.3 (Pseudorandom distribution)**.** A distribution D on $\{0,1\}^m$ is called (S,ϵ)*-pseudorandom* if for any circuit C on m input gates and size at most S,

$$\left| \Pr_{r \sim U_m}[C(r) = 1] - \Pr_{r \sim D}[C(r) = 1] \right| \le \epsilon.$$

When the above happens, we say that D "fools" all circuits of size at most S.

Definition 4.4 (Pseudorandom generator)**.** A function $G : \{0,1\}^* \to \{0,1\}^*$ is said to be a $m(\ell)$*-pseudorandom generator* if for $r \in \{0,1\}^\ell$,

(a) $G(r) \in \{0,1\}^{m(\ell)}$,

(b) $G(r)$ can be computed in $2^{O(\ell)}$ time, and

(c) the distribution over $\{0,1\}^{m(\ell)}$ which takes a uniformly random element s of $\{0,1\}^\ell$ and takes value $G(s)$ is $(m(\ell)^3, 1/10)$-pseudorandom.

The existence of the above would imply that any randomized algorithm in BPP using m random bits and running time m^3 can be simulated by a deterministic algorithm with running time $O(2^{O(\ell)} m^3)$. We merely run the algorithm on $G(r)$ for all $r \in \{0,1\}^\ell$, and output whatever answer (yes or no) occurs more.
If we want to completely derandomize our randomized polynomial-time algorithm to get a deterministic polynomial time algorithm, we want that $m = 2^{\Omega(\ell)}$. In particular, if a pseudorandom generator with $\ell = O(\log m)$ exists, then BPP $=$ P. When this is true, we say that the PRG has "exponential stretch".
Our goal in the next few lectures will be to show that "circuit lower bounds" imply the existence of pseudorandom generators.

Theorem 4.5. There exists a "PRG" with exponential stretch which satisfies only (a) and (c) in the definition.

Proof. Choose a random G – for each $s \in \{0,1\}^\ell$, set $G(s)$ to be a uniformly random string in $\{0,1\}^m$.
Fix a circuit C of size at most m^3. Suppose that $\Pr_{r \sim U_m}[C(r) = 1] = p$, and let

$$B_C := \{r \in \{0,1\}^m : C(r) = 1\}.$$

Note that the random variable

$$X_C := \left| \left\{ s \in \{0,1\}^\ell : G(s) \in B_C \right\} \right|$$

is the sum of 2^ℓ Bernoulli random variables equal to 1 with probability p. By the Chernoff bound,

$$\Pr_G\left[|X_C - \mathbb{E}[X_C]| > \epsilon 2^\ell\right] \leq 2e^{-(\epsilon 2^\ell)^2/3\mathbb{E}[X_C]} = 2e^{-(\epsilon 2^\ell)^2/(3p\cdot 2^\ell)} \leq 2e^{-\epsilon^2 2^\ell/3}.$$

Now, the number of circuits with size at most m^3 is bounded from above by 2^{3m^3}. An application of the union bound yields that

$$\Pr_G[\text{for some } C \text{ of size at most } m^3, |X_C - \mathbb{E}[X_C| > \epsilon 2^\ell] \leq e^{-(\epsilon^2 2^\ell/3)+(\ln 2)(3m^3+1)}.$$

For $\epsilon = 1/10$, $\ell = 4\log m$, and sufficiently large m, the RHS is less than 1, so there exists some G that satisfies (a) and (c) in the definition of a PRG. ■

4.1.2. Lecture 15

In the previous lecture, we had said the following (after defining what a PRG is).

Theorem 4.6. If a $2^{\Omega(\ell)}$-PRG exists, BPP = P.

Note that a 2^ℓ-PRG does not exist. Indeed, if it did, we could design a circuit that is 1 precisely at each point in $\{0,1\}^{2^\ell}$ that is mapped to by the PRG, and is 0 everywhere else.
While no PRGs are known that fool *all* circuits of size bounded by $m(\ell)^3$, there are PRGs known under more specific conditions on the circuit. For example, we can get a PRG that fools any randomized algorithm that is log-space.[7] It is also known that there exist (non-trivial) PRGs which fool constant-depth circuits.

Now, what are *circuit lower bounds*? We had remarked in the previous lecture that they imply the existence of PRGs.

Definition 4.7 (Worst-case hardness)**.** For $f : \{0,1\}^n \to \{0,1\}$, its *worst-case hardness* $H_{\text{worst}}(f)$ is the largest number S such that for any circuit of size at most S, there exists some $x \in \{0,1\}^n$ such that $C(x) \neq f(x)$.

We cannot compute the function using a circuit of size any smaller than its worst-case hardness. The implementation of the truth table yields that the worst-case hardness of any function is at most about $O(2^n)$.

Does there exist any function which is actually this hard? There are 2^{2^n} functions from $\{0,1\}^n \to \{0,1\}$, and there are (about) 2^S circuits of size at most S. Consequently, some functions do require an S of at least about $2^n/n$.
However, no such function is explicitly known – this is another huge open question! In fact, the hardest explicit function we know has worst-case hardness just $3n - o(n)$. As mentioned at the beginning of this section, we can use (worst-case) hard functions to design good pseudorandom generators.

Definition 4.8 (Average-case hardness)**.** For $f : \{0,1\}^n \to \{0,1\}$, its *average-case hardness* $H_{\text{avg}}(f)$ is the largest number S such that for any circuit of size S,

$$\Pr_{x\sim U_n}\left[C(x) \neq f(x)\right] > \frac{1}{2} + \frac{1}{S}.$$

Note that we can trivially get a circuit that is equal to f with probability $\geq 1/2$, either set it as the constant 0 or the constant 1 (depending on which value f takes more often).
Clearly, the average-case hardness of any function is at least the worst-case hardness.

[7] This does not make sense in our current framework, but it is possible to modify the definition of PRGs appropriately. In this setting, we do not have exponential stretch, but we can go from $\Omega(\log^2 m)$ to m. The question of whether RL = L is a huge open question.

4.2. Lectures 15, 16: Average-case hardness to derandomization

4.2.1. Lecture 15 (contd.)

Theorem 4.9 (Nisan-Wigderson)**.** If (for sufficiently large n) there exists a function computable in time $2^{O(n)}$ with $H_{\text{avg}}(f) \geq 2^{2n/3}$, then there exists a $(2^{\ell/45})$-PRG and in particular, BPP = P.

This links the worlds of algorithms (in the time complexity of f), circuits, and derandomization.

Before moving to the proof of the above, let us try to go from ℓ to $\ell + 1$.

Proposition 4.10. Let $f : \{0,1\}^\ell \to \{0,1\}$ be such that $H_{\text{avg}}(f) \geq \ell^4$. Then, G defined by

$$G(r) = (r_1, \ldots, r_\ell, f(r)) = (r, f(r))$$

is a PRG.

That is, we would like to say that the output of the function cannot be predicted for a given input. The above merely says that unpredictability implies indistinguishability.

Theorem 4.11 (Yao's Theorem)**.** Let D be a distribution on $\{0,1\}^m$. Suppose that for any i and any circuit of size $2S$,

$$\Pr_{y \sim D}[C(y_1, \ldots, y_i) = y_{i+1}] < \frac{1}{2} + \epsilon.$$

Then, for any circuit B of size S,

$$\left| \Pr_{y \sim D}[B(y) = 1] - \Pr_{y \sim U_m}[B(y) = 1] \right| < m\epsilon.$$

4.2.2. Lecture 16

Proof of Yao's Theorem. We shall show the contrapositive of the statement. Let B be a circuit of size S such that

$$\Pr_{y \sim D}[B(y) = 1] - \Pr_{y \sim U_m}[B(y) = 1] \geq m\epsilon.$$

We remove the modulus because we can consider the probability that it is 0 otherwise. Define a sequence of distributions $D_0, D_1, \ldots, D_m$ as follows, where D_i is obtained by drawing x from D, and then replacing the first i coordinates with draws from U_m. That is, a draw is $(y_1, \ldots, y_i, z_{i+1}, \ldots, z_m)$, where $y \sim D$ and $z \sim U_m$. Note that $D_0 = U_m$ and $D_m = D$. Let

$$P_i = \Pr_{r \sim D_i}[B(r) = 1].$$

Because $P_m - P_0 \geq m\epsilon$, there is some i such that $P_i - P_{i-1} \geq \epsilon$. Note that D_i and D_{i-1} differ only at the ith bit. We shall give an algorithm to predict y_i given $y_1, \ldots, y_{i-1}$ for $y \sim D$ and a random choice of $z \sim U_m$. If $B(y_1, \ldots, y_{i-1}, z_i, \ldots, z_m)$, then output z_i, and if it is 0 then output $1 - z_i$. For the sake of succinctness, let $x = (y_1, \ldots, y_{i-1}, z_i, \ldots, z_m)$. Now, the probability of success is

$$\frac{1}{2}\left(\underbrace{\Pr[B(x) = 1 \mid y_i = z_i]}_{P_i} + \underbrace{\Pr[B(x) = 0 \mid y_i = 1 - z_i]}_{(1-\alpha),\text{ say}} \right)$$

We have

$$P_{i-1} = \Pr[B(x) = 1] = \frac{1}{2}\left(\Pr[B(x) = 1 \mid y_i = z_i] + \Pr[B(x) = 1 \mid y_i = 1 - z_i]\right) = \frac{1}{2}(P_i + \alpha).$$

Therefore,

$$\text{probability of success} = \frac{1}{2}(P_i + 1 - \alpha) = \frac{1}{2} + P_i - P_{i-1} \geq \frac{1}{2} + \epsilon.$$

To get the final circuit C, note that on a random choice of $z \sim U_m$ in our "algorithm", we succeed with probability at least $(1/2) + \epsilon$. Therefore, there exists some specific choice which gives a probability of success at least $(1/2) + \epsilon$, which is precisely what we want. ■

Let us now come to the proof of Theorem 4.9. The idea is as follows. We would like to consider a bunch of subsets of $[\ell]$, and apply a hard function f to each of them to get one extra bit to append. In all, the number of bits we append is the number of subsets we choose. If we choose all subsets to be disjoint, then the resulting new bits are completely independent of each other, but we do not get exponentially many new bits. Therefore, we allow some small amount of intersection of the subsets, and thus some small amount of correlation, without compromising the uncorrelation of the new bits by too much.

Definition 4.12. An (l, k, d)-*combinatorial design* is a collection $I_1, \ldots, I_r$ of size k subsets such that for distinct $i, j \in [r]$, $|I_i \cap I_j| \leq d$.

Proposition 4.13. For $k = \ell/30, d = k/3$, there exists an (l, k, d)-design of size at least $2^{d/10} \geq 2^{\ell/900}$.

We do not prove this.

Proof of Nisan-Wigderson. Set $\ell = 900 \log n$, and k as from the above.
Fix some combinatorial design $\mathcal{I} = \{I_1, \ldots, I_n\}$ guaranteed by the above, and let $f : \{0,1\}^k \to \{0,1\}$ be a hard function. Then, the final pseudorandom bits we output are $f(z_{I_r})$ for each $r \in [n]$. For simplicity, denote $f(I_r) = f(z_{I_r})$.
Let f be computable in time $2^{O(k)}$ and $H_{\text{avg}}(f) \geq 2^{2k/3}$. Denote the resulting PRG by $\text{NW}^f_{\mathcal{I}}$. We shall show that $\text{NW}^f_{\mathcal{I}}(U_\ell)$ is $(2^{2k/3}/2, 1/10)$-pseudorandom, which is $(n^{20}/2, 1/10)$-pseudorandom.
Now, we shall use Yao's Theorem, by showing unpredictability instead. That is, we are done if we show that for any circuit C of size at most n^{20},

$$\Pr_{z \sim U_\ell}\left[C(f(z_{I_1}), \ldots, f(z_{I_{i-1}})) = f(z_{I_i})\right] \leq \frac{1}{2} + \frac{\epsilon}{n}.$$

Let $f_j(z) = f(z_{I_j})$ for each j.
Suppose otherwise. Let $z' = z_{[\ell] \setminus I_i}$, and $z'' = z_{I_i}$, so

$$\Pr_{z \sim U_\ell}\left[C(f_1(z', z''), \ldots, f_{i-1}(z', z'')) = f(z'')\right] > \frac{1}{2} + \frac{\epsilon}{n}.$$

Ignore z' for now, fixing their values as something (this will be done in precisely the same way as in Yao's Theorem), abuse notation to denote the new functions by f_j as well. Then,

$$\Pr_{z \sim U_\ell}\left[C(f_1(z''), \ldots, f_{i-1}(z'')) = f(z'')\right] > \frac{1}{2} + \frac{\epsilon}{n}.$$

using this, we get a circuit for f that succeeds with probability at least $(1/2) + \frac{\epsilon}{n}$. Note that each $f_j(z'')$ uses at most d bits. By taking $(i-1)$ trivial circuits of f, which are each of size at most about $d2^d$, we get a circuit for f of size $d2^d 2^{d/10} + 2^{2d}/2 \leq 2^{2d}$, contradicting the hardness of f. ■

4.3. Lecture 17: Worst-case hardness to average-case hardness

4.3.1. Lecture 17

Now, we would like to convert a function that is worst-case hard to some other function that is average-case hard.

Definition 4.14. Given $0 < \rho < 1$ and a function $f : \{0,1\}^n \to \{0,1\}$, define $H^{\rho}_{\text{avg}}(f)$ to be the largest S such that for any circuit C of size S,

$$\Pr_{x \sim U_m}[f(x) = C(x)] < \rho.$$

In particular, we have $H_{\text{worst}}(f) = H^1_{\text{avg}}(f)$.

Given a function $f : \{0,1\}^n \to \{0,1\}$ with worst-case hardness $H_{\text{worst}}(f)$, we may view it as a string $\mathsf{TruthTable}(f)$ in $\{0,1\}^{2^n}$. For any circuit S of size less than $H_{\text{worst}}(f)$, we have $\mathsf{TruthTable}(f) \neq \mathsf{TruthTable}(C)$. This idea of using a few inequalities to generate a large number of inequalities is reminiscent of error correcting codes – maybe we can convert the truth table to an element of $\{0,1\}^{2^{2n}}$, say, using an encoder E of relative distance $1/4$, and our new average-case hard function f' is defined by $\mathsf{TruthTable}(f') = E(\mathsf{TruthTable}(f))$.
However, this trick does not allow us to show that *all* circuits differ from f' on a constant fraction! It only does so for circuits whose truth table is equal to some image of the encoding function.
Can we do something in the backward direction? Given a circuit B on $2n$ inputs of size $S^{1/5}$ (say) that agrees with f' significantly (on a $\frac{1}{2} + \frac{1}{S}$ fraction), is it possible to construct a small circuit C on n inputs of size S that agrees with f everywhere?

Let us do something slightly weaker, and suppose that B agrees with f' on a 0.9 fraction. This essentially asks us to decode the code. However, the size of the input string to the decoding algorithm is 2^{2n}, so we need a very fast decoding algorithm.
In usual decoding, we look at the entire corrupted encoded string, and try to retrieve the entire message. In *local decoding*, we read a small part of the corrupted string to recover some portion of the original message. Indeed, finally, we do not want to know the values of f at *every* point, only at the input point x.
We can compute the value of B at this small part, and use it to recover the specific bit corresponding to $f(x)$.

Definition 4.15 (Local decoding)**.** Let $E : \{0,1\}^N \to \{0,1\}^M$ be a encoder that runs in $\text{poly}(N)$. A *local decoder* for handling ρ errors is an algorithm D such that given random access to a string $y \in \{0,1\}^M$ with $\Delta(y, E(x)) \leq \rho$ for some $x \in \{0,1\}^n$ and an index $j \in [N]$, runs in time $\mathsf{polylog}(M)$ and outputs x_j with probability at least $2/3$.

Theorem 4.16. Suppose we have an encoder E with a local decoder D for handling ρ errors. Further suppose we have functions $f_n : \{0,1\}^n \to \{0,1\}$ in EXP that have worst-case hardness $H_{\text{worst}}(f) \geq S(n)$. Then, there exists $\epsilon > 0$ and $\hat{f}_m \in \mathsf{EXP}$ on m bits such that

$$H^{1-\rho}_{\text{avg}}(\hat{f}_n) \geq (S(2\epsilon m))^{\epsilon}.$$

The proof is exactly as in the preceding paragraphs. We have $(\hat{f}(x))_{x \in \{0,1\}^m} = E((f(x))_{x \in \{0,1\}^n})$ Let $N = 2^n$ and $M = 2^m$. Suppose instead that B is a circuit of size $T = (S(\epsilon m))^{\epsilon}$ such that

$$\Pr_{x \sim U_m}[B(x) = f(x)] \geq 1 - \rho.$$

Now, we have that

$$\Delta((B(x))_{x \in \{0,1\}^m}, (\hat{f}(x))_{x \in \{0,1\}^m}) < \rho.$$

Fix x to be the string in $\{0,1\}^n$ that we wish to obtain the value of (of f). Each time we compute $B(y)$ for some y, we incur a time of T.
We still need to eliminate the randomness in the local decoding algorithm. To do this, modify the encoding algorithm by repeating it enough times that the error probability is less than $1/N^2$. Then, for a fixed f, x, this implies that there exists some *fixed* set of random bits which outputs $f(x)$ correctly for all $x \in [N]$.

4.4. Lecture 18–19: Local decoding

4.4.1. Lecture 18

In this lecture, we elaborate a bit more on local decoding. Recall Reed-Solomon codes from Section 3. Are these locally decodable?
Given a polynomial

$$m = a_0 + a_1 x + \cdots + a_{d-1} x^{d-1}$$

and $(m(\alpha_1), m(\alpha_2), \ldots, m(\alpha_n))$, can we recover a specific a_i by looking at a few of the $m(\alpha_i)$? They do not seem very suitable for local decoding, so let us look at some other codes that are more amenable to this.

Definition 4.17 (Reed-Muller codes)**.** Let $\mathbb{F}$ be a finite field, and $\ell, d \in \mathbb{N}$ such that $|\mathbb{F}| > d$. Also fix $S_1, \ldots, S_\ell \subseteq \mathbb{F}$ The message space of the *Reed Solomon code* $\mathrm{RM}(n, \ell, d)$ is

$$\{p(x_1, \ldots, x_\ell) \in \mathbb{F}[x_1, \ldots, x_\ell] : \deg(p) \leq d\}.$$

A polynomial p is encoded as

$$\mathrm{Enc}(p) = (p(\alpha_1, \ldots, \alpha_\ell))_{\alpha \in (S_1 \times \cdots \times S_\ell)}.$$

In our setting, we fix all the S_i to be $\mathbb{F}$.
That is, the encoding goes from $\mathbb{F}^{\binom{d+\ell}{\ell}}$ to $\mathbb{F}^{|\mathbb{F}|^\ell}$. What is the distance of this code? Given a nonzero polynomial p over ℓ variables of degree at most d, what is the largest number of zeros it can have?

Proposition 4.18. Any polynomial $p \in \mathbb{F}[x_1, \ldots, x_\ell]$ of degree at most d has at most $d|\mathbb{F}|^{\ell-1}$ zeros. That is,

$$\Pr_{\alpha \sim \mathbb{F}^\ell}[p(\alpha) = 0] \leq \frac{d}{|\mathbb{F}|}$$

Proof. Assume wlog that the degree of p is d. The idea is that we will partition $\mathbb{F}^\ell$ into a bunch of "lines" and show that on each line, the probability is at most $d/|\mathbb{F}|$. For $\alpha \in \mathbb{F}^\ell, r \in \mathbb{F}^\ell$, consider the line

$$L_{\alpha,r} = \{\alpha + tr : t \in \mathbb{F}\}.$$

We shall show that for some clever choice of r, the polynomial does not become the zero polynomial on this line for any α. Restricted to this line, the function becomes a polynomial in t. We want to show that this is a nonzero polynomial

$$p(\alpha_1 + tr_1, \ldots, \alpha_\ell + tr_\ell).$$

in t. Let P_d be the degree d part of P, and note that the coefficient of t^d in this polynomial is $P_d(r_1, \ldots, r_\ell)$, *independent of* α! Further, P_d cannot be identically zero on $\mathbb{F}^\ell$ because this would imply that the degree of p is less than d (this uses the fact that $|\mathbb{F}| > d$). Therefore, the polynomial is nonzero for some choice of r. This means that the univariate polynomial is nonzero, so has at most $d/|\mathbb{F}|$ zeros, and we are done. ■

Are the Reed-Muller codes locally decodable? Let us change our perspective slightly, changing the message space from the coefficients to the evaluations of $\mathbb{F}$ at some $\binom{\ell+d}{d}$ (fixed and specific) points – there exists a choice of such points which uniquely determines the polynomial.
In the absence of errors, this makes local decoding trivial. What do we do in the presence of errors?
Suppose we want to evaluate the polynomial at some point β given the evaluations at all points (with an ϵ fraction of errors). If we manage to come up with some line through β that has relatively few errors, then we can use Reed-Solomon decoding on this line to compute what $p(\beta)$ is precisely. Suppose that we choose this line randomly. Then, the expected number of errors is

$$\begin{aligned}\mathbb{E}_{\text{random line } \ell \text{ through } \beta}[\text{number of corruptions on } \ell] &= \mathbb{1}_{\text{error at } \beta} \frac{1}{(|\mathbb{F}|^\ell - 1)/(|\mathbb{F}|-1)}(\text{number of errors not at } \beta) \\ &\leq 1 + \epsilon|\mathbb{F}|.\end{aligned}$$

Therefore, by a Markov argument,

$$\Pr_\ell[\ell \text{ has less than } 3(\epsilon|\mathbb{F}|+1) \text{ errors}] \geq \frac{2}{3}$$

and we are done.

In all, we choose a random line through β, apply Reed-Solomon coding on this line, then use the resultant polynomial to compute $p(\beta)$.
Here, the local decoding algorithm runs in $O(|\mathbb{F}|)$ time, which we wish to be $\mathsf{polylog}(|\mathbb{F}|^\ell)$. For sufficiently large ℓ ($\Omega((|\mathbb{F}|/\log|\mathbb{F}|)^\delta)$ for some constant $\delta > 0$), this is indeed true.

When we try to convert this to the binary setting however, one major issue pops up. We can of course view $\mathbb{F}$ as a string over $\{0,1\}^{\log|\mathbb{F}|}$, but in this case the notion of "error" changes. An ϵ fraction corruption means that an ϵ fraction of the *bits* are corrupted, not points in $\mathbb{F}^\ell$. Indeed, an ϵ fraction of bits being corrupted means that an $\epsilon\log|\mathbb{F}|$ fraction of the points in $\mathbb{F}^\ell$ could be corrupted.
We would like a coding scheme over the binary alphabet that can tolerate a constant fraction of errors, and Reed-Muller codes do not seem to satisfy this.

4.4.2. Lecture 19

In the last lecture, we saw that the relative distance of the Reed Muller code was $1 - d/|\mathbb{F}|$, when viewed as a code on alphabet $|\mathbb{F}|$. When viewed as a code on alphabet $\{0,1\}$ however, this goes to $(1-d/|\mathbb{F}|)/\log|\mathbb{F}|$. This issue of the relative distance being $o(1)$ cannot be fixed even by changing $\mathbb{F}, \ell, d$.
To fix this, we shall do the following: for each element of $\mathbb{F}$ (each coordinate when viewed as a code on alphabet $\mathbb{F}$), we shall replace it with another codeword, possibly larger. That is, if we encode it as $x \in \mathbb{F}^{|\mathbb{F}|^\ell}$ under the Reed-Muller code, we encode each x_i as another element $\{0,1\}^t$, where t will end up being $\log|\mathbb{F}|$.

This second code is the *Walsh-Hadamard code*, defined as follows. The encoding is a function $\mathrm{WH} : \{0,1\}^k \to \{0,1\}^{2^k}$, where for each $S \subseteq [k]$, we have $(\mathrm{WH}(x))_S = \bigoplus_{i \in S} x_i$.
We claim that the relative distance of this code is $1/2$. Indeed, if we change r bits, all coordinates corresponding to subsets that contain an odd number of these r bits will change.
Further, it turns out that this optimal.

Proposition 4.19. For any $\delta > (1/2)$, there exists n_0 such that no code with more than 2^{n_0} codewords has relative distance Δ.

Proof sketch. Let us just look at the case where the code is over $\{0,1\}^{n_0}$, and suppose instead that we have a mapping $f : \{0,1\}^{n_0} \to \{-1,1\}^m$ with relative distance $\Delta > 1/2$. Note that $\langle f(x), f(y)\rangle < 0$ for any distinct $x, y \in \{0,1\}^{n_0}$. The result follows by bounding the number of such vectors from above. ∎

In addition, the Walsh-Hadamard code is indeed locally decodable. Given x and some corruption of $\mathrm{WH}(x)$, we can consider sets of the form T and $T \cup \{i\}$, where $i \notin T$. Adding (XORing) the two should give x_i in the absence of corruption. When there is corruption, we can just choose a bunch of random T and perform this same operation, taking the majority finally. The probability that both T and $T \cup \{i\}$ are uncorrupted is at least $1 - 2\rho$, so for $\rho < (1/2)$, we are fine.

In conclusion, our final code is $\mathrm{WH}(\mathrm{RM}(x))$.[8] Here, WH is a mapping from $\{0,1\}^{\log|\mathbb{F}|} \to \{0,1\}^{|\mathbb{F}|}$. The relative distance of this code is $(1/2)(1 - d/|\mathbb{F}|)$, which is $\Theta(1)$ for appropriate $d, |\mathbb{F}|$! We can handle an error fraction of about $\rho \approx \Delta/2 \approx (1/4)$.
One interesting thing is that due to the previous proposition, we cannot even do better than $1/4$ using a coding theory-based proof like this.
Now, we have gone from exponential H_{worst} to exponential $H_{\text{avg}}^{1-\rho}$, which in the limiting case is $H_{\text{avg}}^{3/4}$. How do we go from this to H_{avg}? We do not delve into the details of this, but the main result used is the following.

Theorem 4.20 (Yao's XOR Lemma)**.** Given a function $f : \{0,1\}^n \to \{0,1\}$, define the function $\hat{f} : \{0,1\}^{nk} \to \{0,1\}$ defined by

$$f(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_k) = f(\overline{x}_1) \oplus f(\overline{x}_2) \oplus \cdots \oplus f(\overline{x}_k),$$

where each $\overline{x}_i$ is in $\{0,1\}^n$.
If $\delta > 0$ and $\epsilon > 2(1-\delta)^k$,

$$H_{\text{avg}}^{(1/2)+\epsilon}(\hat{f}) \geq \frac{\epsilon^2}{400n} H_{\text{avg}}^{1-\delta}(f).$$

Given a function with exponentially large $H_{\text{avg}}^{1-\delta}$, making ϵ appropriately exponentially small does the job (around $H_{\text{avg}}^{1-\delta}(f)^{-1/3}$).
Alternatively, one way to go directly from H_{worst} to H_{avg} is to use *local list decoding* for the Reed-Muller and Walsh-Hadamard combination we saw earlier in the lecture.

Therefore, if we have a function that has exponential worst-case hardness, $\mathsf{BPP} = \mathsf{P}$!

[8] mildly abusing notation to mean that we apply WH on a coordinate-by-coordinate basis to $\mathrm{RM}(x)$.

§5. Some more randomized algorithms

5.1. Bipartite matching

5.1.1. Lecture 20

The problem we shall study now is that of finding a perfect matching in a bipartite graph $G = (X, X, E)$. That is, we have two copies of a set X with all edges between the two copies.
This is a problem as old as computer science itself, and quite recently an almost-linear time algorithm for the above has been found [**?**]
We shall give an algebraic algorithm due to Mulmuley, Vazirani, Vazirani [MVV87]. It is rather simple, and is also parallelizable. Consider the *biadjacency matrix* A_G of G, with rows and columns indexed by X and $(A_G)_{uv} = 1$ if uv is an edge and 0 otherwise. Note that the X used to index the rows and columns are different (choose which one is used for rows arbitrarily).
Recall the determinant

$$\det(M) = \sum_{\sigma \in S_n} \operatorname{sign}(\pi) \prod_{i=1}^{n} M_{i,\sigma(i)}$$

and permanent

$$\operatorname{perm}(M) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} M_{i,\sigma(i)}.$$

Suppose that the vertex sets X in our bipartite graph are $[n]$.
Note that any perfect matching essentially corresponds to a permutation of $[n]$ such that there is an edge between i and $\sigma(i)$ for every $i \in [n]$, that is, $(A_G)_{i,\sigma(i)} = 1$ for all i. Due to this, we can also assign a sign to any perfect matching.

Clearly, the number of perfect matchings is then just $\operatorname{perm}(A_G)$. On the other hand,

$$\det(A_G) = \sum_{M \text{ is a perfect matching}} \operatorname{sign}(M).$$

Therefore, if G does not have a perfect matching, $\det(A_G) = 0$. The converse is clearly not true as seen by $K_{2,2}$, which has biadjacency matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Similarly, the determinant is 0 if any two vertices have the same neighbour set.
How do we change something to make the converse hold true (with high probability)? The idea is rather simple, and involves changing by biadjacency matrix by replacing each element with a random integer from $\{1, 2, \ldots, 2n\}$.
Call this new (random) matrix M_G. We claim that in this new setting, $\det(M_G) \neq 0$ with probability at least $1/2$.
Indeed, consider the determinant polynomial in n^2 variables, which is

$$\det \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix}.$$

Note that this is a degree n polynomial.

Lemma 5.1 (Polynomial Identity Lemma)**.** Let p be a polynomial in m variables of degree d. Then,

$$\Pr_{\alpha \sim \{1,2,\ldots,2d\}^m} [p(\alpha) \neq 0] \geq \frac{1}{2}.$$

We have already seen a proof of this back in Lecture 18 (in the case where coefficients are rational), where we worked with $\mathbb{F}_p$ instead. Indeed, something being nonzero modulo p implies nonzeroness in $\mathbb{R}$.
An alternative proof is by induction on the number of variables.

Using this lemma in our setting, we see that $\det(M_G) = 0$ with probability at most $1/2$, so we are done. Further, we can use this algorithm to actually find a perfect matching. For $i \in [n]$, assuming that the perfect matching has the edge $1i$, check if the remaining part of the graph has a perfect matching. If yes, find a perfect matching on it (recursively). Otherwise, increment i.

Now, can we come up with a *parallel* algorithm for constructing a perfect matching? Assuming we have polynomially many machines that run independently, is it possible to determine a perfect matching rapidly, say in constant or logarithmic time?

In the simple case where we have a *unique* perfect matching, this is quite simple by running m many machines parallelly, each computing a determinant of the graph excepting the vertices in an edge e. If for a given e the determinant is nonzero, the edge must be in the matching.
The algorithm we shall see has its idea centered around the above observation.

Suppose that we can assign weights to the edges $w : E \to \mathbb{Z}$ such that the minimum weight perfect matching is unique, where the weight of a matching M is

$$w(M) = \sum_{e \in M} w(e).$$

We then alter the biadjacency matrix A_G so that the edge e's entry is $2^{w(e)}$ instead of 1. Then,

$$\det(A_G) = \sum_{\text{perfect matchings } M} \text{sign}(M) 2^{w(M)}.$$

Note that due to the uniqueness, the above determinant is nonzero! It cannot be cancelled by any sum of higher weight matchings (Why?). After that, for each edge, decrement the weight by one and see if the minimum weight has now decreased. If it has, this edge must be part of the minimum weight perfect matching.
All that remains is to find a weight assignment such that there is a unique minimum weight perfect matching. It turns out that a random weight assignment does the trick. This is not immediately clear, because if we assign weights in $\{1, 2, \ldots, n^2\}$, say, then despite there being possibly exponentially many matchings, the minimum weight one is unique.

5.1.2. Lecture 21

Lemma 5.2 (Isolation Lemma, [MVV87])**.** Let E be a set of m elements and $\mathcal{S} \subseteq 2^E$ an arbitrary family of subsets of E. Independently and uniformly randomly assign to each element of E a weight in $\{1, 2, \ldots, N\}$. Then,

$$\Pr\left[\mathcal{S} \text{ has a minimum weight set}\right] \geq 1 - \frac{m}{N},$$

where the weight of a set is the sum of the weights of the elements in it.

We get the desideratum in the context of matchings on setting E to be the set of edges and $\mathcal{S}$ to be the collection of perfect matchings.

Proof. Let $E = \{e_1, \ldots, e_m\}$. Split $\mathcal{S}$ into two parts $\mathcal{S}_0, \mathcal{S}_1$, where $\mathcal{S}_0 = \{T \in \mathcal{S} : e_1 \in T\}$ and $\mathcal{S}_1 = \mathcal{S} \setminus \mathcal{S}_0$. Let us look at the event E that there is a minimum weight set that contains e_1 and a minimum weight set that does not contain e_1. This means that the minimum weight set in $\mathcal{S}_0, \mathcal{S}_1$ are equal.

What happens if we fix the weights of all elements other than e_1? The minimum weight in $\mathcal{S}_1$ is determined, and the minimum weight in $\mathcal{S}_{\prime}$ is just equal to some fixed quantity plus the weight of e_1. In particular, there is at most one value of $w(e_1)$ such that the two minimum weights are equal. Therefore, $\Pr[E] \leq 1/N$. In general, taking the union bound, we have

$$\Pr[\text{there exist two min wt sets}] = \Pr\left[\bigcup_{i\in[m]} \text{there exist min wt sets containing } e_i \text{ and not containing } e_i\right] \leq \frac{m}{N}.$$

■

Later, ***** proved that the above is in fact true with $\left(1 - \frac{1}{N}\right)^m$ instead. Note that the above is true if we replace the set weights are drawn from with any set of size N, so perturbing about $\log N$ bits ensures a unique solution.

The isolation lemma has several surprising applications, for example that UNIQUE-SAT[9] is NP-hard.

We next look at derandomization. We cannot derandomize the isolation lemma in all its generality, but we can for specific families that have some structure.
For example, this is very easy for spanning trees and it suffices to assign distinct weights to all edges. Our main goal is that of derandomizing *bipartite perfect matching*. We will only be able to derandomize it to $O(\log^2 n)$ random bits unfortunately, which is equivalent to giving $n^{O(\log n)}$ weight assignments with the assurance that one of them gives a minimum weight matching.

The high-level view of the proof is the following.
The weight construction is done in $\log n$ rounds. We start off with some huge (exponentially large) family of perfect matchings. We then come up with some weight function such that the set of perfect matchings of minimum weight is comparatively smaller. We then come up with another weight function (with about $\log n$ bits) to break ties among these minimum weight perfect matchings and make the set even smaller. Further, we ensure that the older non-minimum weight matchings do not suddenly enter this family by appending the bits of the new weight function to the bits of the previous weight function. Each of these bit sequences we append are $\log n$ bits, and because there are $\log n$ rounds we end at $\log^2 n$ bits in all.
As before, let the edges be $e_1, \ldots, e_m$.
For starters, observe that if $w(e_i) = 2^i$ for all i, then all subsets have distinct weights.
Let M_1, M_2 be two minimum weight perfect matchings. Observe that $M_1 \cup M_2$ is a union of cycles (and possibly isolated edges contained in both M_1, M_2). Further, each cycle in $M_1 \cup M_2$ has zero "alternating weight". This is the difference of the sum of all "odd" edges in the cycle and the sum of all "even" edges in the cycle. If we instead had that the M_1 sum was greater than the M_2 sum, we could switch out the edges in the cycle in M_1 for edges in the cycle in M_2 to get a matching of weight strictly less than that of M_1, yielding a contradiction.

Lemma 5.3. Let $E' \subseteq E$ be the union of all minimum weight perfect matchings. Then, each cycle in $G = (V, E')$ is has zero alternating weight.

Corollary 5.4. If w is a weight assignment such that a cycle C has nonzero alternating weight, then the union of minimum weight perfect weight matchings (with respect to w) does *not* contain C.

The above corollary is the key idea. For a suitable weight assignment on a cycle, we can get rid of at least one edge in the cycle, and this ensures that all matchings containing that edge are rid of. Our goal then is to maximize the number of edge-disjoint cycles in the graph.

Proposition 5.5. Let $C_1, \ldots, C_k$ be an arbitrary collection of cycles. Then, for some $j \in \{1, 2, \ldots, m^2 k\}$, the weight function defined by $w(e_i) = 2^i \pmod{j}$ for each i assigns a nonzero alternating weight to every cycle C_r.

[9] This is SAT, except that we know that if there is a satisfying assignment, it is unique.

Proof. Given a cycle C and a weight assignment w, let $w_{\pm}(C)$ be the alternating weight of C under w. We would like to show that for some j, j is not a factor of $w_{\pm}(C_1)w_{\pm}(C_1)\cdots w_{\pm}(C_k)$. This product is at most 2^{m^2k}. Recalling that the lcm of the first n numbers is greater than 2^n for sufficiently large n, we have that 2^{m^2k} is less than the lcm of $[m^2k]$, so there is some number in $[m^2k]$ that does not divide 2^{m^2k}. ■

References

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, Mar 1987.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), sep 2008.