CS 761 : Derandomization and Pseudorandomness

Amit Rajaraman

Last updated September 26, 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 $O(n^{2.373\ldots})$.
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,

$$\begin{aligned}\Pr\left[z_1r_1 + \cdots + z_nr_n = 0 \mid z_n \neq 0\right] &= \Pr\left[r_n = -\frac{z_1r_1 + \cdots + z_{n-1}r_{n-1}}{z_n} \mid z_n \neq 0\right] \\ &\leq \max_{r_1,\ldots,r_{n-1}} \Pr\left[r_n = -\frac{z_1r_1 + \cdots + z_{n-1}r_{n-1}}{z_n} \mid z_n \neq 0, r_1, \ldots, r_{n-1}\right]\end{aligned}$$

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 $\{0, 1, \ldots, q-1\}$. It is not difficult to see that

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

Inspired by this, we can reduce randomness as follows. Choose x randomly from $\{0, 1, \ldots, 2n-1\}$, and set $r = (1, x, x^2, \ldots, x^{n-1})$. 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 $X_1, \ldots, X_n$ 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. 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 $X_1, \ldots, X_k$, 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 $a_1, \ldots, a_m$ 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,

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

Proof. We have

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

■

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

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

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 $|\{i \in [n] : x_i \neq y_i\}|$. 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

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

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 $\mathrm{Rate}(\mathcal{C}_n) \to r_0$?

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 2.1. 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 2.2. 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 2.5. 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,

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

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 $\text{wt}(x) \leq n/10d$. Let $S \subseteq [n]$ be the subset of L such that $v \in S$ iff $x_v \neq 0$. By Lemma 2.2, 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

```
isPath(s, t, k = 2^ℓ)
    // Outputs whether there is a path between s and t of length at most k
    if ℓ = 0 then
        return yes iff s, t are adjacent
    foreach v ∈ V(G) do
        return yes iff isPath(s, v, 2^{ℓ-1}) and isPath(s, v, 2^{ℓ-1})
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

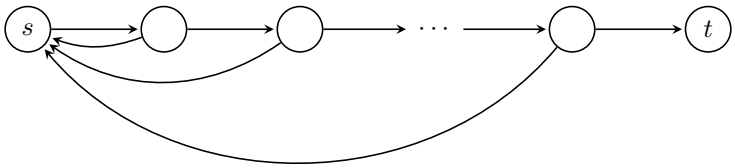
Input: An graph G, and vertices $s, t \in V(G)$
Output: Connectivity of s, t

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

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

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 2.6. 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 2.3. 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 $G \circledcirc H$.

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 $G \circledcirc H$ 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 $G \circledcirc H$, so we are done. ■

What happens to the eigenvalue of $G \circledcirc H$? 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 2.4. It is true that

$$\gamma(G \circledcirc H) \geq \gamma_1 \gamma_2^2.$$

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 $G_k^2 \circledcirc H$.

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 2.4 without much trouble that $\lambda(G \circledcirc H) \leq \lambda_1 + 2\lambda_2$.
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 2.4, 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

```
G_0 ← regularize(G)
for 1 ≤ k ≤ O(log n) do
    G_{k+1} ← G_k^2 Ⓩ H
```

First, make G a regular graph. G_0 is an $(n, D^2, 1 - \frac{1}{\mathrm{poly}(n)})$ 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 $1/\mathrm{poly}(n)$ to a constant in $O(\log n)$ steps!
The only thing that remains is to prove Lemma 2.4.

Lemma 2.5. 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,

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

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 2.4. 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 $G \textcircled{z} H$ is just $\tilde{B}\tilde{A}\tilde{B}$!
Now, denote by J the matrix that has all elements $1/n$.
Now, use Lemma 2.5 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

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

completing the proof. ■

2.3. Lecture 9: Expander graphs

2.3.1. Lecture 9

Definition 2.7. 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] $\|AB\| \leq \|A\|\|B\|$. This is obvious because $\|ABx\| \leq \|A\|\|Bx\| \leq \|A\|\|B\|\|x\|$.

Definition 2.8 (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 $d^{O(\log n)} = \text{poly}(n)$ paths to explore. Therefore, there is a *deterministic* polynomial time algorithm to determine connectivity of expander graphs.

Definition 2.9 (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 2.10 (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 2.6 (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. 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 $H_n \coloneqq P_2^{\otimes n}$. 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. 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 2.7 (Expander Mixing Lemma)**.** Let G be a (n,d,λ)-expander. Then, for any $S,T \subseteq V$,

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

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 2.8. 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,

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

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

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

Definition 2.11 (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 3.1. 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 $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$, 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 $d_H(\mathrm{Enc}(g), r) < (n - k)/2$ 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 3.2. If there exists a polynomial $h \in \mathbb{F}[x]$ of degree at most $k - 1$ such that $d_H(\mathrm{Enc}(h), r) < (n - k + 1)/2$, 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 $\{\alpha_1, \ldots, \alpha_n\}$ is at least the number of agreements between $\mathrm{Enc}(h)$ 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 3.3 (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 $\mathrm{poly}(q, n, \Delta)$.

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

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

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 $\mathrm{agr}(h, \mathcal{S}) \geq t$. 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 $\mathrm{poly}(d, \log q)$.[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 $R(x) \coloneqq$

[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 $\mathrm{agr}(h, \mathcal{S}) \geq t$. 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 3.2. 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 $\mathrm{agr}(h, \mathcal{S}) \geq t$. 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.

Definition 3.3 (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 $\mathbb{F}^r$, where

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

The rate of this code is approximately k/nr, which is worse than before. The distance however, jumps up to $n - \frac{k-1}{r}$!
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 r, multiplicity codes can be efficiently decoded for agreement $(1+\epsilon)\frac{k}{r}$. We can get arbitrarily close to the (hard) bound!
Recent state-of-the-art expander graphs are constructed using multiplicity codes!

§4. Hardness v. Randomness

4.1. Lecture 13: Hardness

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 time $O(n^c)$ for some constant c such that

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

Here, we mean that r is a random
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 time $O(n^c)$ for some constant c 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,
$$\left| \Pr_{r \sim U_m} [B(x, r) = \text{yes}] - \Pr_{r \sim D} [B(x, r) = \text{yes}] \right| \leq \epsilon.$$
One neat way to think about algorithms *with input* is circuits.
In fact, any deterministic algorithm can be viewed as a family $(C_n)_{n \geq 1}$ 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 $r_1, \ldots, r_m$.

Definition 4.3 (Pseudorandom distribution on $\{0,1\}^m$)**.** 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,
$$\left| \Pr_{r \sim U_m} [C(r) = 1] - \Pr_{r \sim D} [C(r) = 1] \right| \leq \epsilon.$$

Definition 4.4 (Pseudorandom generator)**.** Let $G = (G_\ell)_{\ell \in \mathbb{N}}$ be a family of functions, where $G_\ell : \{0,1\}^\ell \to \{0,1\}^{m(\ell)}$ (for some appropriate increasing function m). G is said to be a *pseudorandom generator* if for each ℓ,

1. G_ℓ can be computed in $O(2^\ell)$ time, and
2. 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 $O(2^{2\ell} m^3)$. 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 $\ell = O(\log m)$.

References

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