CS 761 : Derandomization and Pseudorandomness

Amit Rajaraman

Last updated August 29, 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. Lecture 7: 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

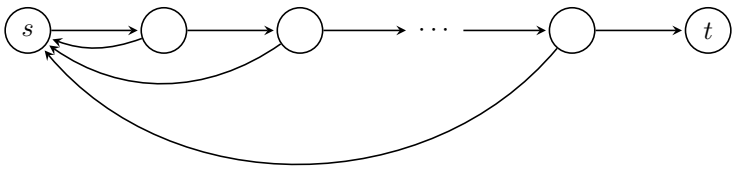
Input: An graph G, and vertices $s, t \in V(G)$
Output: Connectivity of s, t

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?).

References

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