CS 761 : Derandomization and Pseudorandomness

Amit Rajaraman

Last updated August 18, 2022

Contents

§1. Introduction

1.1. Lecture 1: Matrix Multiplication

We begin with a question.

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

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

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

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

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

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

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

1.2. Lectures 3–4: Pairwise Independence

1.2.1. Lecture 3

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.2.2. Lecture 4

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

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

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

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. Lecture 4: Applications

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.

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, $\mathbb{E}[m] = 1/(k + 1)$.