Lecture 16: Proof of the Schwartz-Zippel lemma, determinants and permanents

Anup Rao

May 16, 2024

The Schwartz-Zippel Lemma turns out to be quite useful for randomized algorithms. We stated the lemma, last time, this time we give its proof:

Lemma 1. *Let $p(x_1, \ldots, x_n)$ be a polynomial of degree d, such that p is not the 0 polynomial. Let S be any set of numbers, and let $a_1, \ldots, a_n$ be n random numbers drawn from S. Then $\Pr[p(a_1, \ldots, a_n) = 0] \leq d/|S|$.*

Proof We prove the lemma by induction on n. When $n = 1$, the theorem follows from the fact that any non-zero degree d polynomial in one variable has at most d roots. Thus $p(a) = 0$ only when a is a root, which happens with probability at most d.

For the general case. Let us write the polynomial in the form

$$p(x_1, \ldots, x_n) = x_n^{\ell} \cdot q(x_1, \ldots, x_{n-1}) + r(x_1, \ldots, x_n),$$

where here r is a polynomial in which the degree of x_n is at most $\ell - 1$. So we simply gather all the terms which have maximum degree in x_n.

Now let E_1 be the event that $p(a_1, \ldots, a_n) = 0$, and let E_2 be the event that $q(a_1, \ldots, a_{n-1}) = 0$. Then we have that

$$\begin{aligned}
\Pr[E_1] &= \Pr[E_1 \wedge E_2] + \Pr[E_1 \wedge \neg E_2] \\
&= \Pr[E_2] \cdot \Pr[E_2|E_1] + \Pr[\neg E_2] \cdot \Pr[E_1|\neg E_2] \\
&\leq \Pr[E_2] + \Pr[E_1|\neg E_2].
\end{aligned}$$

By induction, since q is a degree $d - \ell$ polynomial, $\Pr[E_2] \leq (d - \ell)/|S|$. Since after $x_1, \ldots, x_{n-1}$ are fixed in $\neg E_2$, we have that $p(a_1, \ldots, a_{n-1}, x_n)$ is a non-zero polynomial of degree ℓ, we have that $\Pr[E_1|\neg E_2] \leq \ell/|S|$. Thus $\Pr[E_1] \leq d/|S|$. ■

Using polynomials to give fast algorithms for matching

Given a bipartite graph with n vertices on the left and n vertices on the right, a *perfect matching* is a set of n disjoint edges. It is a classical problem in graph algorithms to figure out if a graph has a perfect matching. Here we present a randomized algorithm using the Schwartz-Zippel lemma.

Recall that the determinant of an $n \times n$ matrix is a polynomial of the form

$$\mathsf{det}(M) = \sum_{\pi} \mathsf{sign}(\pi) \prod_{i=1}^{n} M_{i,\pi(i)},$$

where here the sum is over all permutations $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$, and $\mathsf{sign}(pi)$ is one of ± 1.

The algorithm is as follows. First define the matrix of variables:

$$M_{i,j} = \begin{cases} x_{i,j} & \text{if } (i,j) \text{ is an edge of the graph,} \\ 0 & \text{otherwise.} \end{cases}$$

Observe that $\mathsf{det}(M)$ is the 0 polynomial if and only if the graph does not have a perfect matching. So, the algorithm simply sets $x_{i,j}$ to be a random number in $\{1, 2, \ldots, 100n\}$ and evaluates $\mathsf{det}(M)$. If $\mathsf{det}(M) = 0$ we conclude the graph has no perfect matching. Otherwise we conclude that the graph does have a perfect matching. Since the degree of the polynomial is at most n, the probabilty that this algorithm makes an error is at most $1/100$ by the Schwartz-Zippel Lemma.

It turns out that the determinant can be computed in $O(\log^2 n)$ circuit-depth, so it can be computed extremely fast in parallel. This gives a very fast parallel time algorithm for this classic problem.

The Permanent

The permanent of an $n \times n$ matrix M is defined to be $\sum_{\pi} \prod_{i=1}^{n} M_{i,\pi(i)}$, where the sum is taken over all permutations $\pi : [n] \rightarrow [n]$.

The permanent is important because it is a complete function for the class #**P**:

Definition 2. *A function $f : \{0,1\}^n \rightarrow \mathbb{N}$ is in* #**P** *if there exists a polynomial p and a poly time machine M such that*

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$$

For example, in #**P** one can count the number of satisfying assignments to a boolean formula, which is potentially much harder than just determining whether the formula is satisfiable or not. One can show that any such problem can be reduced in polynomial time to computing the permanent of a matrix with $0/1$ entries. On the other hand, the permanent itself can be computed in #**P**. Thus the permanent is #**P**-complete.