Lecture 8: $\mathbf{ZPP} = \mathbf{RP} \cap co\mathbf{RP}$*, determinants and introduction to interactive proofs.*

Anup Rao

November 15, 2023

We began the lecture by proving:

Theorem 1. $\mathbf{ZPP} = \mathbf{RP} \cap co\mathbf{RP}$.

Proof Suppose $f \in \mathbf{ZPP}$, via a randomized algorithm M whose expected running time is $t(n)$. Consider the algorithm that simulates M for $10t(n)$ steps, and outputs 0 if the simulation halts. Then clearly, the algorithm only makes an error if the correct answer is 1. On the other hand, the probability that running time of M exceeds $10t(n)$ is at most $1/10$ (or else the expected running time would exceed $t(n)$. Thus we obtain an **RP** algorithm. The same idea (reversing the roles of 0 and 1) gives a $co\mathbf{RP}$ algorithm.

For the other direction, suppose f has an **RP** algorithm M_1 and a $co\mathbf{RP}$ algorithm M_0. Then on input x consider the algorithm that alternatively runs $M_0(x), M_1(x), M_0(x), \ldots$ until either $M_1(x)$ outputs 1, or $M_0(x)$ outputs 0. If $M_1(x) = 1$, then it must be that $f(x) = 1$. Similarly if $M_0(x) = 0$, it must be that $f(x) = 0$. In any case, one of these two algorithms will verify the value of x in an expected constant number of runs. ■

Next, we discussed polynomial identity testing. The notes for this are in the previous week's lecture.

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 $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, 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 $\det(M)$ is the 0 polynomial if and only if the graph has a perfect matching. So, the algorithm simply sets $x_{i,j}$ to be a random number in $\{1, 2, \ldots, 100n\}$ and evaluates $\det(M)$. If $\det(M) = 0$ 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.

Interactive proofs

One way to define **NP** is via the idea of a proof system. **NP** is the set of functions f for which there is a polynomial time verifier algorithm V such that given any x with $f(x) = 1$, there exists a prover P that can prove to the verifier that $f(x) = 1$ by providing a polynomial sized witness w for which $V(x, w) = 1$, yet if $f(x) = 0$, no such prover exists.

What happens if we allow the verifier to have a longer *interactive* conversation? Presumably, giving the verifier the ability to adaptively ask the prover questions based on his previous responses should give the verifier more power, and so allow the verifier to verify the correctness of the value for a larger set of functions. In fact, this does *not* give the verifier additional power: for if there is such an interactive verifier V^I for verifying that $f(x) = 1$, we can design a non-interactive verifier that does the same job. The new verifier will demand that the prover provide the entire transcript of interactions between V^I and a convincing prover. The new verifier can then verify that the transcript is correct, and would have convinced V^I. Thus, if f has an interactive verifier, then $f \in \mathbf{NP}$.

The story is more interesting if we allow the verifier to be randomized. We say that $f \in \mathbf{IP}$ if there is a polynomial time randomized verifier V such that

Completeness For all x, if $f(x) = 1$, there is an oracle P such that $\Pr_r[V^P(x, r) = 1] \geq 2/3$.

Soundness For all x, if $f(x) = 0$, for every oracle P, $\Pr_r[V^P(x, r) = 1] \leq 1/3$.

Since any prover can be simulated in polynomial space, if $f \in \mathbf{IP}$, then $f \in \mathbf{PSPACE}$. The algorithm for f can just try all possible sequences of messages from the prover until it finds a sequence of messages that convinces the verifier, if such a sequence exists.

Theorem 2. $\mathbf{IP} \subseteq \mathbf{PSPACE}$.

It is easy to check that allowing the prover to be randomized does not change the model.

We shall eventually prove that $\mathbf{IP} = \mathbf{PSPACE}$ (and so $\mathbf{IP}$ is potentially much more powerful than $\mathbf{NP}$).

Example: Graph non-Isomorphism

Two graphs on n vertices are said to be *isomorphic* if the vertices of one of the graphs can be permuted to make the two equal.

Consider the problem of testing whether two graphs are *not* isomorphic: the boolean function f such that $f(G_1, G_2)$ is 1 if and only if G_1 is not isomorphic to G_2. $f \in co\mathbf{NP}$, since the prover can just send the verifier the permutation that proves that they are isomorphic. We do not know if $f \in \mathbf{NP}$, but it is easy to prove that $f \in \mathbf{IP}$.

Here is the simple interactive protocol:

1. The verifier picks a random $i \in \{1, 2\}$.
2. The verifier randomly permutes the vertices of G_i and sends the resulting graph to the prover.
3. The prover responds with $b \in \{1, 2\}$.
4. The verifier accepts if $i = b$.

If G_1, G_2 are not isomorphic, then any permutation of G_i determines i, so the prover can determine i and send it back. However, if G_1, G_2 are isomorphic, then the graph that the prover receives has the same distribution whether $i = 1$ or $i = 2$, thus the prover can guess the value of i with probability at most $1/2$. Repeating the protocol several times, the verifier can make the probability of being duped by a lying prover exponentially small.

A protocol for counting satisfying assignments

We continue to exhibit the power of interaction by showing how it can be used to solve any problem in $\mathbf{PSPACE}$. Recall that the problem of computing whether a totally quantified boolean formula is true is complete for $\mathbf{PSPACE}$, so it will be enough to give an interactive protocol that verifies that such a formula is true.

As a warmup, let us consider the case when we are given a formula of the type $\exists x_1, \ldots, x_n \phi(x_1, \ldots, x_n)$ and want to *count* the number of satisfying assignments to this formula. Since the permanent is complete for **#P**, we can reduce this counting problem to the computation of the permanent, and then use the interactive protocol from the last lecture, but let us be more direct.

As in the protocol for the permanent, we shall leverage algebra. Since polynomials are much nicer to deal with than formulas, let us try to encode the formula ϕ using a multivariate polynomial. Here is a first attempt at building such an encoding gate by gate:

- $x \wedge y \to xy$.
- $\neg x \to 1 - x$.
- $x \vee y \to x + y - xy$.

This encoding gives us a polynomial g_ϕ that computes the same value as the formula ϕ, however it is not clear that g_ϕ can be computed in polynomial time. The problem is the encoding for $\vee$ gates, which could potentially double the size of the polynomial obtained in each step. Instead, we use the more clever encoding:

- $x \vee y \to 1 - (1 - x)(1 - y)$.

This allows us to obtain a polynomial g_ϕ which can be written down in time polynomial in the size of ϕ.

Then the task of counting the number of satisfying assignments to ϕ reduces to computing $\sum_{x \in \{0,1\}^n} g_\phi(x)$. Following the ideas used in the protocol for the permanent, here is a protocol for a verifier that checks that $\sum_{x \in \{0,1\}^n} g_\phi(x) = k$.

1. Ask the prover for a prime $2^{2n} > p > 2^n$, and check that it is correct. Reject if $k < p$. All arithmetic is henceforth done modulo p.
2. If $n = 1$, check the identity by computing it.
3. If $n > 1$, ask the prover for the degree n polynomial
$$f(X) = \sum_{x_2, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n).$$
4. Check that $f(0) + f(1) = k \mod p$.
5. Pick a random element $a \in \mathbb{F}_p$ and recursively check that
$$f(a) = \sum_{x_2, x_3, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \ldots, x_n)$$

For the analysis, note that $f(X)$ is indeed a degree n polynomial, since there are at most n gates in the formula ϕ. Thus if

$$\sum_{x \in \{0,1\}^n} g_\phi(x) = k,$$

an honest prover can convince the verifier with probability 1.

If $\sum_{x \in \{0,1\}^n} g_\phi(x) \neq k$, then the if the prover succeeds, it must be that

$$f(X) \neq \sum_{x_2,\ldots,x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n),$$

for if the prover is honest, he will be caught immediately.

Since $f(X), \sum_{x_2,\ldots,x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n)$ are both degree n polynomials, we have that

$$\Pr_a \left[f(a) = \sum_{x_2,\ldots,x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \ldots, x_n) \right] \leq n/p,$$

so with high probability, the prover is left with trying to prove an incorrect statement in the next step. By the union bound, the probability that the prover succeeds in any step is at most $n^2/p \ll 1/3$ for large n.