6.5420 Randomness and Computation February 3, 2026

Lecture 1: Polynomial Identity Testing

Lecturer: Ronitt Rubinfeld *Scribe: Morgan Prior*

1 Introduction & Notation

While many problems which are solvable by efficient randomized algorithms also have fast deterministic algorithms, polynomial identity testing is different in that the best known randomized algorithm is strictly more efficient than the best known deterministic one. This lecture introduced the problem of polynomial identity testing for both single-variable and multi-variable polynomials. We saw an efficient randomized algorithm for this problem, as well as applications to the "Person on the Moon" communication problem and to the bipartite matching problem.

The motivating question is: given two polynomials, say $P(x) = (x+1)^2$ and $Q(x) = x^2 + 2x + 1$, are they equal? In this concrete example, by expanding P or factoring Q, it is easy to tell that they are equal.

Consider instead $P(x) = (x+3)^{38}(x-4)^{83}$ and $Q(x) = (x-4)^{38}(x+3)^{83}$. Because the degree here is much higher, comparing these polynomials after expanding them is inefficient. Fortunately, in the factored form provided, we can give a short justification that $P \neq Q$: consider the input $x = 0$ and observe that $P(0)$ is negative while $Q(0)$ is positive. When the two polynomials are not given in factored form like this, such a short justification is not always possible. This challenge motivates the more general problem of polynomial identity testing:

Problem 1. *Given two polynomials P and Q, each with degree at most d, is $P(x) = Q(x) \quad \forall x$?*

Problem 2. *Given a polynomial R with degree at most d, is $R(x) = 0 \quad \forall x$?*

By taking $R = P - Q$, which clearly has degree at most $\max(\deg(P), \deg(Q)) \leq d$, we can observe that Problem 1 reduces to Problem 2. In the following sections, we see efficient, randomized algorithms for solving Problem 2.

Throughout the notes, we use $P \equiv Q$ to denote that P and Q are identical polynomials; that is, $P(x) = Q(x) \quad \forall x$. Similarly, we use $R \equiv 0$ to mean that a polynomial R is identically zero; that is, $R(x) = 0 \quad \forall x$. We use $i \in_D S$ where D is a probability distribution to mean that i is a random D-distributed draw from the set S, and $i \in_U S$ to mean i is a uniformly distributed draw. We use $[n]$ to denote the set $\{1, \ldots, n\}$.

2 Univariate Polynomial Identity Testing

Our randomized algorithms make use of the following theorem:

Theorem 3. *If $R \not\equiv 0$ and has $\deg \leq d$, then R has $\leq d$ roots.*

Remark. Theorem 3 actually holds over any field. In particular, it holds for the field $\mathbb{Z}_q$, where q is a prime, which is the set $\{0, 1, \ldots q-1\}$ with the operation of addition mod q. This fact will be useful when doing the Human on the Moon application in Section 2.3.

2.1 Deterministic Algorithm

Theorem 3 suggests a simple deterministic algorithm for deciding whether $R \equiv 0$:

1. Pick $d+1$ distinct points $x_1, \ldots, x_{d+1}$ in the domain of R.
2. If $\forall i$, $R(x_i) = 0$, output "$R \equiv 0$."
3. Otherwise (i.e., we found i s.t. $R(x) \neq 0$), output "$R \not\equiv 0$."

Clearly, this algorithm is correct on all inputs R. Indeed, if R really is identically zero, we will always terminate at Step 2. If R is not identically zero, by the Pigeonhole Principle and the fact that $\deg(R) \leq d$, at least one of the x_i will not be a root, and we will always terminate at Step 3. Despite its correctness, the algorithm requires $d+1$ evaluations of R, which is inefficient when d is large. This motivates the faster algorithm described in the next subsection.

2.2 Randomized Algorithm

1. Pick $2d$ distinct points $x_1, \ldots x_{2d}$ in the domain of R.
2. Do the following k times:
   (a) Draw $i \in_U [2d]$.
   (b) If $R(x_i) \neq 0$, output "$R \not\equiv 0$."
3. Output "$R \equiv 0$."

Remark. Note that in the randomized algorithm above, we do not make any assumption about the points $x_1, \ldots, x_{2d}$ other than distinctness. However, the indices i must be drawn uniformly and independently of each other.

Observe that our algorithm's error is one-sided; it is always correct when $R \equiv 0$. However, when $R \not\equiv 0$, there is some chance that in all k rounds we select an x_i that is a root of R, and thus output the wrong result. If $R \not\equiv 0$, using the Fundamental Theorem of Alegbra to upper bound the number of roots, we have that on any given round,

$$\Pr_{i \in [2d]}[R(x_i) = 0] = \frac{\#\text{ roots}}{2d} \leq \frac{d}{2d} = \frac{1}{2}.$$

Since the x_i are drawn independently, we get the following upper bound on the error probability of the whole algorithm:

$$\Pr[\text{error}] = \Pr[\text{choose a root on all } k \text{ iterations}] \leq \frac{1}{2^k}.$$

To achieve an error probability less than some fixed parameter δ, we can select $k = O(\log 1/\delta)$.

2.3 Application: Human on the Moon

Suppose you have some binary string $w = w_0 \ldots w_n$ and you are communicating with someone on the moon who has binary string $w^* = w_0^* \ldots w_n^*$. You want to know if $w = w^*$. The naive solution is for one participant to send their whole string, but this costs $O(n)$ bits of communication, which is expensive if n is large.

We can use polynomial identity testing to solve this problem by constructing two degree n polynomials: $P(x) = w_n x^n + \cdots w_1 x + w_0$ and $P^*(x) = w_n^* x^n + \cdots w_1^* x + w_0^*$. We have that $w = w^*$ iff $P \equiv P^*$.

2.3.1 Randomized Algorithm (a first attempt)

Consider the following randomized algorithm for the Human on the Moon problem:

1. You (the person on Earth) pick random $r_1, \dots r_k \in [2n]$ and send $(r_1, P(r_1)), \dots (r_k, P(r_k))$ to the moon person.
2. The moon person evaluates P^* on the same inputs and checks if $P^*(r_i) = P(r_i)$ for all $i \in [k]$.

A second look at this proposed algorithm reveals its ineffectiveness: when evaluating $P(x)$, we can get values on the order of $O(n^n)$, which would require $O(n \log n)$ bits to send to the person on the moon. This is worse than the naive strategy of sending the entire string.

2.3.2 Randomized Algorithm, revised

To remedy the problems with our first attempt, we will do all computations modulo some prime $q \geq 2n$:

1. You (the person on Earth) pick random $r_1, \dots r_k \in [2n]$ and compute $p_i = P(r_i) \pmod q$ for all $i \in [k]$.
2. You send $(r_1, p_1), \dots (r_k, p_k)$ to the moon person.
3. The moon person evaluates whether $P^*(r_i) \mod q = p_i$ for all $i \in [k]$.

Fact 4 (Bertrand's Postulate)**.** *For any n, there exists a prime $q \in (2n, 4n)$.*

By Fact 4, q can be chosen to be at most $4n$, which means it takes at most $O(\log n)$ bits to represent. Then both the r_i and p_i take at most $O(\log n)$ bits to present, and since $k = O(1)$, the entire algorithm requires $O(\log n)$ bits of communication, outperforming the naive approach.

3 Multivariate Polynomial Identity Testing

We can ask the same question for multivariate polynomials:

Problem 5. *Given $R(x_1, x_2, \dots, x_n)$ with degree at most d, is $R \equiv 0$?*

When we refer to degree in Problem 5, we mean *total degree*:

Definition 6 (Total degree)**.** *Given a polynomial $P(x_1, x_2, \dots, x_n)$, the* total degree *of P is the maximum over all terms in P of the sum of the degrees of the x_i's in the term.*

Example 7. *Let $P(x, y, z) = 2xy + 3z^3 + 4xyz^2$. The degree of the first term is $1+1 = 2$, the degree of the second term is 3, and the degree of the third term is $1 + 1 + 2 = 4$. The total degree of P is* $\max(2, 3, 4) = 4$.

The multivariate case has some new challenges:

- There exist multivariate polynomials R such that $R \not\equiv 0$ but R has many roots. For example, define $R(x, y) = xy$. When x and y are both non-zero, $R(x, y) \neq 0$. However, if, say $x = 0$, we get a distinct root of R for each possible value of y.
- If we have n variables and degree d, we can have $\binom{n}{d}$ distinct degree-d terms.

Using the following lemma, we can construct a randomized algorithm in this case analogous to the one in Section 2.2:

Lemma 8 (DeMillo–Lipton–Schwartz–Zippel)**.** *Let $R(x_1, \dots, x_n)$ be a polynomial of total degree d such that $R \not\equiv 0$. Let S be a set of elements in the domain of R. Pick $x_i \in_U S$ $\forall i \in [n]$ then* $\Pr[R(x_1, \dots x_n) = 0] \leq \frac{d}{|S|}$.

The lemma can be proved by induction on the degree.

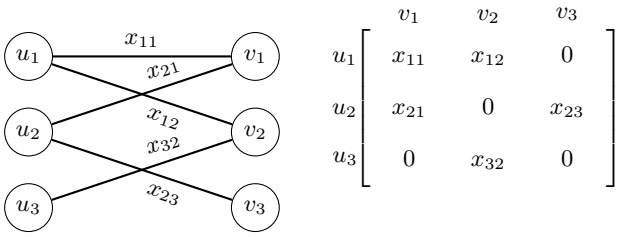

Figure 1: An example of a bipartite graph G and the corresponding matrix A_G, labeled based on the edges of G.

3.1 Bipartite Perfect Matching

Recall that a bipartite graph is a graph with no odd cycles. Multivariate polynomial identity testing can actually be used to determine a given bipartite graph has a perfect matching. This problem is solvable in polynomial time deterministically by using network flows.

We define the matrix A_G such that $a_{i,j} = \begin{cases} X_{ij} & \text{if } (i,j) \in G \\ 0 & \text{otherwise} \end{cases}$. Figure 3.1 shows an example of a bipartite graph and its associated matrix.

Let S_n denote set of permutations of 1 to n. A permutation is a bijective function $\sigma \in S_n$. Notice that we can associate each permutation σ with a matching in G, by thinking of the pairs $(i, \sigma(i))$ as the edges in the matching.

Definition 9 (determinant, permanent)**.** *Given a matrix A, the* determinant *of A is defined as:*

$$\det(A) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

The permanent *of A is similar, but with the summands unsigned:*

$$\text{per}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$$

Notice that for an n-vertex graph G, if σ corresponds to a perfect matching in G, then the term $\prod_{i=1}^{n} a_{i,\sigma(i)}$ in $\text{per}(A_G)$ equals one. This tells us that $\text{per}(A_G)$ is equal to the *number* of perfect matchings in G. However, the permanent is significantly harder to compute than the determinant (in terms of computational complexity), so we would like to be able to say something about the presence of a perfect matching using the determinant instead. It turns out that we can:

Lemma 10. *G has a perfect matching if and only if* $\det(A_G) \neq 0$.

Proof. If there is no perfect matching, then for each permutation σ, at least one edge $(i, \sigma(i))$ must be absent from G. Hence the entry $a_{i,\sigma(i)}$ in A_G is zero, and the whole product $\prod_{i=1}^{n} a_{i,\sigma(i)}$ will be equal to zero. This will be true of each term, so the sum $\sum_{\sigma \in S_n} \text{sign}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)} = \det(A_G) = 0$.

On the other hand, if there is a perfect matching, then there is a permutation σ so that the edges $(i, \sigma(i))$ are all present and thus the entries $a_{i,\sigma(i)}$ are all equal to 1. So $\det(A_G) \neq 0$ because it contains the nonzero term $\text{sign}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$. □

Now to tell whether G has a perfect matching, we just need to be able to compute $\det(A_G)$. The challenge is that ordinarily we compute the determinant of a matrix whose entries are constants, and in A_G, some of the entries are variables. To compute the determinant when there are variables,

we use the following clever trick: observe that $\det(A_G)$ is a polynomial with total degree at most n (the number of vertices), so we can do polynomial identity testing on it by evaluating it with random choices for the variables X_{ij}.