6.895 Randomness and Computation February 6, 2008

Lecture 1

Lecturer: Ronitt Rubinfeld *Scribe: Shubhangi Saraf*

1 Introduction

Today we'll discuss a classical example of how randomness helps in algorithms. We'll present a randomized algorithm for the problems of Polynomial Zero Testing and Polynomial Identity Testing and then give two very nice applications of these results.

2 The problems

The problem of *Polynomial Identity Testing* (PIT) is the following. Given two polynomials p and q in n variables, we want to determine if they are the same, i.e. is $p(x_1, x_2, \ldots, x_n) = q(x_1, x_2, \ldots, x_n)$ for all $x_1 \ldots x_n$. For example, consider

$$(x_1+x_2)(x_3+x_4)^{40}(x_5^2+x_6)\stackrel{?}{\equiv}(x_1-x_2)(x_3-x_4)^{40}(x_5^2+x_6)+(x_1+2x_2)(x_3-x_4)^{40}(x_5^2+x_6)+x_1x_3x_5^2+x_2x_6x_3^{40}.$$

We would like to figure out if the polynomial on the left is identically equal to the polynomial on the right. The problem with opening up the brackets and expanding is that we could possibly get an exponential (in the degree of the polynomial) number of terms. For instance, if there are n variables and the degree of d, there could be more that $\binom{n}{d}$ terms!

Consider the following related problem of *Polynomial Zero Testing*. Given a polynomial $p(x_1, \ldots, x_n)$, we want to determine if it's identically 0, i.e. is $p(x_1, x_2, \ldots, x_n) = 0$ for all $x_1 \ldots x_n$.

Observe that both the above problems are equivalent. Clearly Polynomial Zero Testing is a special case of Polynomial Identity Testing when q is the zero polynomial. In the other direction, to determine if $p(x_1, x_2, \ldots, x_n) = q(x_1, x_2, \ldots, x_n)$, instead consider $p'(x_1, x_2, \ldots, x_n) \doteq (p - q)(x_1, x_2, \ldots, x_n)$. Then $p' \equiv 0$ iff $p \equiv q$, and hence we can instead test if p' is identically 0.

Before we discuss the above problems, let us clarify some notions. Assume that the *domain is a field* such as $\mathbb{R}$ or $\mathbb{Z}_p$. For example, if the field is $\mathbb{Z}_7$, then

$$(x+3)^2 \equiv x^2 + 6x + 9 \equiv x^2 + 6x + 2 \pmod 7.$$

The *degree of a univariate polynomial* is the highest exponent of a term in the polynomial. For instance, the degree of $x^{10} + x^3 + 1$ is 10. The *total degree of a multivariate polynomial* is the max over all terms of the sum of degrees in the term. This is the notion we'll tend to use most of the time unless specified otherwise. The *maximum degree of a multivariate polynomial* is the maximum over all terms of the degree of the maximum degree variable in that term. For example, if the polynomial is $xy^2 + x^2$, then the total degree is 3, and the maximum degree is 2.

3 Algorithm for Polynomial Zero Testing

Assume a polynomial p in n variables and of degree d is given as a black-box oracle. The oracle takes as input $\bar{x} = (x_1, \ldots, x_n)$ and outputs $p(\bar{x})$.

Deterministic Algorithm for the Case when p is Univariate. Plug in any $d+1$ distinct values to the black-box. If all are 0, then output "$\equiv$0". Else, output "$\not\equiv$0".

The above algorithm makes $O(d)$ evaluations. It works since a non-zero polynomial of degree at most d can have at most d roots.

Randomized Algorithm for the Univariate Case. The above observation also implies that if the domain field size is greater than $2d$, then at most $d/|F| \leq 1/2$ of the fraction of all field elements are zeroes of the polynomial. This suggests the following randomized algorithm. Pick an element uniformly at random from a field of large enough size and plug it in. If it outputs 0, then output "≡0". Else, output "≢0". The algorithm works in $O(1)$ evaluations. The behavior of the algorithm is as follows. If $p \equiv 0$, the algorithm outputs '$\equiv 0'$. If $p \not\equiv 0$, Pr[algorithm outputs "≢0"] $\geq 1/2$. This same idea is generalized to give an algorithm that works in the multivariate case.

Randomized Algorithm for the Multivariate Case. Observe that a multivariate polynomial can have infinitely many roots - for example: $p(x, y) = x \cdot y$ over a field of infinite size has infinitely many roots. However, the same kind of test that works in the univariate case works in this case too as long as the field is of large enough size, and a random value is picked for each x_i.

Let us first set up some notation that we'll use for the rest for the course.

- $x \in_R S$ denotes "Pick x uniformly from S".
- $x \in_U S$ denotes "Pick x uniformly from S".
- $x \in \mathcal{D}$ denotes "Pick x according to distribution $\mathcal{D}$".
- $x \in_{\mathcal{D}} S$ denotes "Pick x according to distribution $\mathcal{D}$ on set S".

The following is the algorithm that works in the multivariate case (needs $|F| \geq 2d$):

1. Pick $S \subseteq F$ arbitrarily such that $|S| \geq 2d$.
2. Pick $x_1, x_2, \ldots, x_n \in_R S$.
3. If $p(x_1, \ldots, x_n) = 0$, output "≡0"; else output "≢0".

Again, the algorithm needs just $O(1)$ evaluations of p on numbers of size $O(\log d)$.This is shown be the below claim, which can be proved by induction on n, with the univariate case as the base case.

Claim 1 *If the polynomial $p \not\equiv 0$, then* $\Pr[p(x_1, \ldots, x_n) = 0] \leq \frac{d}{|S|}$.

The behavior of the algorithm is as follows. If $p \equiv 0$, the algorithm outputs "≡0". If $p \not\equiv 0$, Pr[algorithm outputs "≢0"] $\geq 1/2$. This probability can be improved by either repeating the algorithm multiple times, or picking a bigger field size.

4 Can we derandomize PIT?

Assume that the polynomial is given as an arithmetic circuit. Kabanets and Impagliazzo proved that if we can derandomize polynomial identity testing, then one of the two statements given below must be true:

1. NEXP is not contained in P/poly.
2. Permanant is not computable by polynomial-size arithmetic circuits.

Hence derandomizing PIT would be a very powerful result.

5 Applications of PIT

5.1 The man on the moon problem

Assume Alice on the earth has a string $\bar{a} = a_1 a_2 \ldots a_n$ and Bob on the moon has a string $\bar{b} = b_1 b_2 \ldots b_n$. They want to decide if $\bar{a} = \bar{b}$.

One way of doing this would be for Alice to transmit $\bar{a}$ all the way to the moon. This would take n bits of transmission.

Alternatively, Alice could view $\bar{a}$ as the coefficients of a degree n univariate polynomial $p = \sum a_i x^i$ over a field of size at least $2n$. Then, pick arbitrarily $S \subseteq F$ such that $|S| \geq 2n$, pick $x \in_R S$, and send x and $\bar{a}(x)$ to Bob (the man on the moon). He checks if $\bar{a}(x) = \bar{b}(x)$. If yes, they conclude that $\bar{a} = \bar{b}$. Else they're different.

By our previous analysis, the above algorithm indeed works. If $\bar{a} = \bar{b}$ then they will always conclude that they are equal, and if $\bar{a} \neq \bar{b}$, then with probability at least a half, they would find that out. In the above randomized algorithm, only $O(\log n)$ bits are needed to be transmitted.

5.2 Bipartite Matching

We'll first introduce some preliminaries. A *bipartite graph* $G = (V, E)$ is one in which the set of vertices V can be partitioned into two sets S and T such that all the edges in G go between S and T, and there are no edges with both endpoints lying in the same set. A *matching* M is a subset of the set of edges E such that no two edges in M share a vertex. A *perfect matching* is one that has an edge adjacent to each vertex.

A natural question that arises in graph theory is that given a graph, decide if it has a perfect matching. The problem of actually finding a perfect matching in a graph that has one, can be solved using network flows. The result we'll show here is not as strong, but is a first step in the direction of getting the best algorithm for finding a bipartite matching. We'll show how to use Polynomial Zero Testing to give a randomized algorithm for determining if a given graph has a perfect matching.

Given a graph $G = (V, E)$, the *Frobenius matrix* $A_G = [a_{ij}]$ of the graph is a matrix whose entries are variables or zeroes. In particular, $a_{ij} = x_{ij}$ if $(i, j) \in E$, and $a_{ij} = 0$ otherwise.

Claim 2 *A graph G has a perfect matching iff* $\det(A_G) \neq 0$.

Proof

$$\det(A_G) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i\sigma_i}$$

Here S_n denotes the set of all permutations of 1 through n. Observe that every permutation represents a possible matching between the two vertex sets. Also, $\prod_{i=1}^{n} a_{i\sigma_i} \not\equiv 0$ if and only if all the edges corresponding the the perfect matching represented by the permutation σ are present in G. Since every other term has a different combination of variables, the non-zero terms don't get canceled out. This completes the proof of the above claim. ■

Since the determinant of A_G is a multivariable polynomial of degree n, it can be tested by the Polynomial Zero Testing algorithm to see if it's identically zero or not, and hence to determine if the graph G contains a perfect matching.

The above idea was used by Lovász to determine if a perfect matching exists in a graph. Since then, it has been shown that similar approaches can be extended to the non-bipartite case as well. For instance, Mucha and Sankowski (FOCS 2004) showed that one can find a maximum matching in general graphs in time $O(n^\omega)$, which is the time required for matrix multiplication. Nick Harvey at MIT (FOCS 2006) showed a simpler algorithm with the same running time.