Lecture 5 – Polynomial Identity Testing

Michael P. Kim

18 April 2017

1 Outline and Motivation

In this lecture we will cover a fundamental problem in complexity theory – polynomial identity testing (PIT). We will motivate our study of identity testing by showing a fast, parallelizable algorithm for testing for a perfect matchings in bipartite graphs that uses PIT as a primitive. The known polynomial-time algorithm for PIT is randomized; in fact, we will see that finding a fast deterministic algorithm for PIT would represent a major breakthrough in complexity theory. Along the way, we will review important concepts from graph theory and algebra.

2 Bipartite Perfect Matching

To motivate our study of polynomial identity testing, we study the problem of testing if a bipartite graph has a perfect matching. A *bipartite graph* is a graph $G = (V = V_L \cup V_R, E)$ where all edges have one vertex in V_L and one vertex in V_R. A *matching* is a set of edges $M \subseteq E$ such that no two edges share a vertex – that is, for all pairs of edges $uv, wx \in M$, $u \neq w, u \neq x, v \neq w, v \neq x$. We say that a graph has a *perfect matching*, if there is a matching such that all vertices participate in some matched edge.

There are a number of polynomial-time algorithms for finding maximum matchings in graphs. In bipartite graphs, it is easy to reduce the problem to a max-flow computation; in general graphs, there are more sophisticated algorithms such as the "Blossom" algorithm. For the special case of testing if a perfect matching exists in bipartite graphs, we can reduce the problem to a more algebraic question[1].

2.1 The Adjacency Matrix and Matchings

Given a graph, consider it's adjacency matrix A where $A_{ij} = 1$ if $ij \in E$ and 0 otherwise. Consider the *permanent* of this matrix defined as

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_i A_{i\sigma(i)}.$$

[1] With more sophisticated techniques, this approach can be made to work for *finding* a perfect matching in general graphs.

It's not too hard to see that the permanent actually counts the number of perfect matchings in a bipartite graph. In particular, note that each perfect matching corresponds to some way to permute the vertices on the left such that they are matched to the vertices on the right. Each such permutation will contribute 1 to the sum and any permutation that corresponds to a non-edge will contribute 0. (Furthermore, because we're going over permutations, we don't have to worry about edges that share endpoints.) Thus, one way we could determine whether a bipartite graph has a perfect matching is to compute the permanent of the adjacency matrix and see if it is non-zero.

There is a problem with this approach. It turns out that counting the number of perfect matchings in a bipartite graph, and thus, computing the permanent, is computationally intractable.

Theorem 2.1 (Valiant '79)**.** Permanent *is* #P*-hard.*

So, in general, we are not going to be able to use this strategy to compute perfect matchings efficiently; nevertheless, the observation that we want to somehow enumerate over permutations of the vertices will be useful.

A syntactically similar quantity to the permanent of a matrix is the *determinant* defined as

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_i A_{i\sigma(i)}$$

The difference in these quantities, syntactically, is just that we add or subtract the value corresponding to certain permutations based on the sign of the permutation. While these quantities look similar, computationally, they are very different beasts! In fact, the determinant of a matrix can be computed deterministically in $O(n^\omega)$ time, where $\omega < 2.373$ is the matrix multiplication constant.

Can we use the determinant to detect whether there is a perfect matching in G? Initially, it seems challenging. In particular, because we've introduced signs into the sum, two non-zero terms could cancel each other out, resulting in a determinant equal to 0 even when there is a perfect matching.

Exercise: Exhibit a graph where the permanent of the adjacency matrix is non-zero but the determinant is zero.

2.2 The Tutte Matrix and Formal Polynomials

The problem with using the adjacency matrix was that we might get cancelations in the determinant, so if $\det(A) = 0$, we can't conclude whether there is or isn't a perfect matching. To address this, we introduce a new matrix, called the Tutte matrix defined as

$$T_{ij} = \begin{cases} x_{ij} & ij \in E \\ 0 & ij \notin E \end{cases}$$

where the set $\{x_{ij}\}$ is a set of distinct formal variables. Now, the determinant is a formal polynomial over the variables $\{x_{ij}\}$. We can evaluate this polynomial over any field, and glean information about it. (Note: The determinant of the adjacency matrix corresponds to the determinant of the Tutte matrix evaluated at $\vec{1}$.) In particular, because every monomial in the determinant is distinct, we observe the following useful fact.

Fact 2.1. *$det(T) \equiv 0$ if and only if G does not contain a perfect matching.*

Here, $p \equiv 0$ means that p is identically zero; that is, every coefficient of every monomial in p is 0. Thus, one way to solve bipartite perfect matching is to reduce it to testing whether a formal polynomial is identitically zero.

3 Polynomial Identity Testing

We say that two polynomials p, q are identical, which we denote $p \equiv q$, if the coefficients in p and q for each monomial are equal[2]. The problem of polynomial identity testing asks, given the description of two polynomials, p and q determine whether $p \equiv q$. Equivalently, we can ask, given a polynomial p', determine whether $p' \equiv 0$ (by setting $p' = p - q$).

When discussing the efficiency of algorithms that compute on polynomials, it is important to consider their representation; for instance, if we represent a polynomial as a list of its coefficient for each monomial, then identity testing is trivially in linear time. But for a polynomial over n variables with degree d, there are $\Omega(n^d)$ monomials. Typically, we want to avoid this exponential dependence on d, so we will assume that we either have oracle access to the polynomial, or that we have a succinct representation of the polynomial (say, in the form of an arithmetic circuit).

There are no known deterministic polynomial-time algorithms for PIT (in fact, not even subexponential – more on this later). There is, however, a very efficient randomized algorithm for PIT. The algorithm hinges on an important lemma about the roots of low-degree polynomials.

Lemma 3.1 (Schwartz-Zippel Lemma)**.** *Let p be a nonzero polynomial on n variables of degree d over a field $\mathbb{F}$. Suppose $S \subseteq \mathbb{F}$; then*

$$\Pr_{\alpha_1,\ldots,\alpha_n \sim S}[p(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|}.$$

Proof. We proceed by induction on n and d. The base case for $n = 1$ is the fact that any univariate polynomial with degree d has at most d roots. The base case for $d = 1$ can be seen through the principle of deferred decisions; fixing the values of $x_1, \ldots, x_{n-1}$ leaves a random equality constraint on x_n.

Now suppose p is a nonzero polynomial on n variables with degree d. Write $p(x_1, \ldots, x_n)$ as follows:

$$p(x_1, \ldots, x_n) = \sum_{i=0}^{d} q_i(x_1, \ldots, x_{n-1}) x_n^i$$

where each q_i is polynomial on $n-1$ variables with degree at most $d-i$. Suppose the only nonzero q is q_0. Then for any assignment to x_n, we are left with a nonzero polynomial on $n-1$ variables and degree d, and the lemma follows by induction. Now suppose there is some $i > 0$ such that

[2]This is not always the same as asking if the polynomials implement the same function (as this depends on the domain we choose to evaluate the polynomial over). In particular, consider the polynomial $x^q - x$ for prime q evaluated over $\mathbb{F}_q$. Fermat's Little Theorem implies this polynomial always evaluates to 0, but it is not identically 0, as two monomials have non-zero coefficients.

q_i is nonzero. Then because q_i has degree at most $d - i$, by induction, the probability a random assignment to $x_1, \ldots, x_{n-1}$ sends q_i to 0 is at most $\frac{d-i}{|S|}$. Further, after fixing the assignment to the first $n-1$ variables, we are left with a degree i univariate polynomial in x_n, which is sent to 0 with probability at most $\frac{i}{|S|}$. Thus, over the entire assignment, the probability is at most $\frac{d-i}{|S|} + \frac{i}{|S|} = \frac{d}{|S|}$. □

Specifically, if we take $\mathbb{F}$ to be a large prime field $\mathbb{F}_q$, and take S to be $\mathbb{F}_q$, then evaluating a nonzero degree d polynomially on a random value in $\mathbb{F}_q^n$ is a zero with probability at most $\frac{d}{q}$. This observation leads to a very natural one-sided error randomized algorithm.

Schwartz-Zippel PIT:
On input $p(x_1, \ldots, x_n)$:

- Select q to be a prime such that $2d^c < q$ (for $c \geq 1$)
- select $\alpha_1, \ldots, \alpha_n \sim \mathbb{F}_q^n$ uniformly at random
- if $p(\alpha_1, \ldots, \alpha_n) \neq 0$: reject
  else: accept

Clearly, if we find an evaluation point that is nonzero, this evaluation certifies that p is not the zero polynomial. If the evaluation is equal to 0, we can bound the probability that this happened by chance for a nonzero polynomial using Schwartz-Zippel. Suppose $p \not\equiv 0$; then by Schwartz-Zippel, the probability that the random evaluation equals 0 is $\leq \frac{d}{2d^c} = \frac{1}{2d^{c-1}} \leq 1/2$. So this gives us an RP algorithm for solving PIT. We can, of course, amplify our probability of correctness by evaluating on many points over the field; alternatively, we can evaluate over a larger field (i.e. increase c), and increase our probability of success on a single evaluation.

3.1 Reducing Bipartite Perfect Matching to PIT

Returning to the question of testing for a perfect matching in bipartite graphs, we can now design an algorithm based on PIT. First of all, note that the degree of the determinant of the Tutte matrix is n. Thus, our algorithm will be as follows. Choose a large prime $q > n^2$ and for every x_{ij}, sample a random field element α_{ij} from $\mathbb{F}_q$. Then, compute the determinant of $T[\alpha]$. Conclude there is a perfect matching if and only if $\det(T[\alpha]) \neq 0$. The analysis of correctness follows directly from the correctness of PIT. Further, this algorithm simply requires evaluating the determinant of a matrix over a prime field, which is efficient both sequentially and in parallel.

Theorem 3.1. *Given a matrix $A \in \mathbb{F}_q^{n \times n}$, computing $\det(A)$ is in* NC.

Corollary 3.1. *Bipartite Perfect Matching is in* RNC.

3.2 Derandomizing PIT?

The randomized algorithm for PIT is very simple and suggests a natural approach to finding a deterministic algorithm – find a fixed set of evaluation points $\left\{\alpha^{(1)}, \ldots, \alpha^{(T(d))}\right\}$, such that for all

nonzero degree d polynomials p in n variables, there exists an $i \in [T(d)]$ such that $p(\alpha^{(i)})$ is nonzero. In fact, it can be shown that such sets exist and can be of polynomial size.

Exercise: As in Problem Set #2, use the probabilistic method to show the existence of such sets.

This fact shows that PIT $\in$ coNP. But despite the fact that, with very high probability, a random set suffices, no known determinsitic constructions for such sets exist, even of *subexponential* size. It turns out that our lack of progress on derandomizing PIT can be blamed on our inability to prove lower bounds more broadly.

Theorem 3.2 (KI '03)**.** *If PIT* $\in \mathsf{NTIME}[2^{o(n)}]$*, then* $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ *or* $\mathsf{Permanent} \notin \mathsf{AlgP/poly}$.

That is, if we could come up with even a slightly better than brute force test set that would certify that a polynomial is acutally the zero polynomial, then we would get new circuit lower bounds. While the general consensus in the complexity community is that nondeterministic exponential time should not have polynomial-sized circuits and that the permanent requires super polynomial-sized circuits, the community also is in general agreement that we are far away from proving such lower bounds. This suggests that derandomizing PIT would not only be a cool algorithmic achievement, but would represent a major breakthrough in our understanding of computational hardness. Of course, if $\mathsf{RP} = \mathsf{P}$, then PIT $\in \mathsf{P}$. Thus, again, we generally believe that there should be some way to derandomize the PIT algorithm, but we currently have no idea how to do this efficiently.