

Lecture 2: Randomized Algorithms

September 9, 2004

Scribe: Peter Pal Zucsek

1 Polynomial Identity Testing

Polynomial Identity Testing is the problem to decide whether two arithmetic expressions are identical. Obviously it's enough to decide whether a single polynomial is identical to 0.

Given a polynomial $p(x_1, x_2, \dots, x_n)$ over some field \mathbb{F} , we want to know if

$$p(x_1, x_2, \dots, x_n) \equiv 0$$

holds.

A natural interpretation could be the following:

Definition for interpretation 1:

$p(x_1, x_2, \dots, x_n) \equiv 0$ if for each $a_1, a_2, \dots, a_n \in D$,

$$p(a_1, a_2, \dots, a_n) = 0$$

where D denotes the domain in which the polynomial is defined.

However, there's another interpretation:

Definition for interpretation 2:

$p(x_1, x_2, \dots, x_n) \equiv 0$ if in the form where the polynomial is written as the linear combination of monomials, each coefficients in that combination is 0.

Note : a monomial is a product $x_1^{d_1} \cdot x_2^{d_2} \cdot \dots \cdot x_n^{d_n}$, $d_i \in \mathbb{N}$.

Let's have a look at some examples where the two different interpretations yield different results.

- **Example 1**

$$p(x_1, x_2, \dots, x_n) = \prod_{i=1}^n (x_i^2 - x_i)$$

The polynomial on the right is not identical to zero according to the second interpretation, however it's clear that $p(a_1, a_2, \dots, a_n) = 0$ for all $a_i \in \{0, 1\}$.

- **Example 2**

For a finite field F , the polynomial

$$p(x) = \prod_{\alpha \in F} (x - \alpha) = x^{|F|} - x$$

vanishes at all points of F but not identical to 0 according to the second interpretation.

Note : for infinite fields, the two concepts yield in the same set of 0-identical polynomials. This is because any non-zero polynomial may have at most finitely many roots.

There’s another important way in which the two interpretations differ: the first is an NP-hard task to evaluate, while the second can be processed in randomized polynomial time. Therefore, from this point on, we will use the second interpretation.

The degree of a monomial $x_1^{d_1} \cdot x_2^{d_2} \cdot \dots \cdot x_n^{d_n}$ is $\sum_{i=1}^n d_i$. The degree of the polynomial is the maximum degree of its monomials.

Example

Let $p(x)$ be the determinant of the Vandermonde-matrix:

$$p(x) = \det \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \dots & x_n^{n-1} \end{vmatrix}$$

Then, $\text{degree}(p(x)) = \frac{n(n-1)}{2}$.

2 Comparison of Arithmetic Formulas

In this section we show a useful application of the Polynomial Identity Testing. Consider the following model of arithmetic formulas, where each operator has two operands (Figure 1):

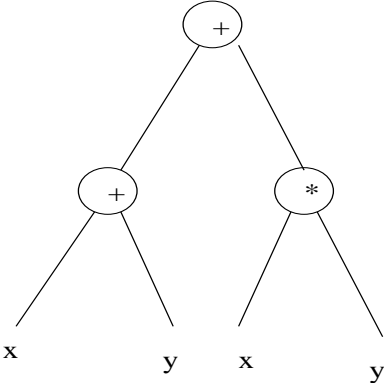


Figure 1: A simple arithmetic formula, $(x + y) + xy$.

The *size* of the formula denotes the number of leaves in the binary graph. The evaluation of such formulas runs in polynomial time of the *size* of the formula on all inputs, hence we do like those algorithms which run P-time in the number of variables.

Note: It’s easy to show that the degree of a polynomial completed by a size s formula is at most s . In contrast, small circuits may compute polynomials whose degree is exponential in the size of the circuit (see, e.g., Figure 2).

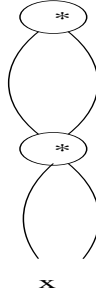


Figure 2: A circuit for x^4 .

Given a pair of arithmetic formulas $a(\bar{x})$ and $b(\bar{x})$, we should decide whether

$$a(\bar{x}) \equiv b(\bar{x}).$$

This can be reduced to the polynomial identity testing as this is equivalent to

$$a(\bar{x}) - b(\bar{x}) \equiv 0$$

which is

$$p(\bar{x}) \equiv 0, \text{ where } p(\bar{x}) = a(\bar{x}) - b(\bar{x}).$$

Our randomized algorithm for solving this problem will be the following:

Input: a (representation of the) polynomial $p(x_1, x_2, \dots, x_n)$ over \mathbb{F} of degree d , where d might be a function of n .

Question: is $p \equiv 0$?

Algorithm: Take any finite set $S \subseteq \mathbb{F}$, $|S| \geq 2d$.

Pick $\alpha_1, \alpha_2, \dots, \alpha_n$ independently, uniformly at random from the elements of S .

if $p(\alpha_1, \alpha_2, \dots, \alpha_n) \neq 0$, then **reject** (since $p \not\equiv 0$), otherwise **accept**.

The correctness of this algorithm follows from the next lemma.

Lemma [Schwartz, Zippel, de Millo, Lipton,...]

Let $p(x_1, x_2, \dots, x_n)$ be any nonzero polynomial over \mathbb{F} of degree d . Let $S \subseteq \mathbb{F}$ be any finite set. Then

$$\Pr [p(\alpha_1, \alpha_2, \dots, \alpha_n) = 0] \leq \frac{d}{|S|},$$

where $\alpha_1, \alpha_2, \dots, \alpha_n$ are chosen independently, uniformly at random from S .

Note: each α_i is in S , not their tuple.

Proof. Our proof makes an induction on the number of variables. For 1 variable, the inequality is trivial as the polynomial p can have d roots at maximum. Although we do not go into details, it's not hard to show that this inequality is also true for multi-variate polynomials (see, e.g., Madhu Sudan's lecture notes on algebra available from the course web page).

Two applications of Polynomial Identity Testing

- Perfect Matching Testing for bipartite graphs

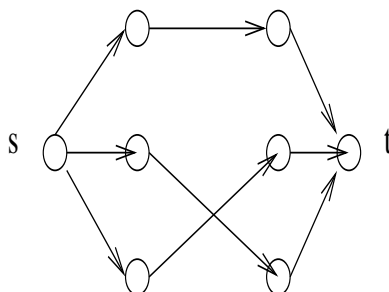


Figure 3: Flow over bipartite graph with capacity-1 edges.

Assume we have a bipartite graph with two parts A and B , each of size n . Add a source vertex s and add edges between s and each members of A . Add a sink vertex t and add edges between t and each members of B . Consider all edges to have capacity 1.

The original graph has perfect matching if and only if there is a flow of value n in the extended graph (see Figure 3). Although we know that the Ford-Fulkerson algorithm solves the maximum flow problem on integer edge capacities in polynomial time, the algorithm itself is very sequential. There is a fast randomized parallel algorithm for this problem, which we describe next.

Idea [Edmonds, Tutte]

Given a bipartite graph $G(V, E)$ on $n + n$ vertices in the form of a matrix where the rows are representing edges from one part while the columns the other:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

where $a_{ij} = \begin{cases} x_{ij} & \text{if } (i, j) \in E \\ 0 & \text{if } (i, j) \notin E \end{cases}$

Hence,

$$\det A = \sum_{\text{permutations } \sigma} (-1)^{\text{sign } \sigma} \prod_{i=1}^n a_{i, \sigma(i)}$$

which is a degree n polynomial in the x_{ij} variables. So we can run our polynomial identity testing algorithm on it.

Claim: $\det A \neq 0 \Leftrightarrow G$ has a perfect matching.

The proof is simple as only those permutations have a non-zero coefficient which correspond to perfect matchings and they can't eliminate each other as they contain different x_{ij} variables.

There is a non-trivial parallel algorithm for computing the determinant of a numeric matrix, which we will not describe here.

• **Primality Testing**

Given an integer n , we should decide whether n is a prime number. We show a simple randomized algorithm given by Agrawal & Biswas. A variant of this algorithm was later derandomized by Agrawal, Kayal and Saxena.

Claim: For any $n \in \mathbb{Z}^+$, $(x + 1)^n \equiv x^n + 1 \pmod{n}$ if and only if n is a prime number.

Proof \Rightarrow :

if $n = p$, prime p ,

$$(x + 1)^p = x^p + \binom{p}{1}x^{p-1} + \binom{p}{2}x^{p-2} + \dots + \binom{p}{p-1}x + 1$$

where p divides all the summands on the right hand side but x^p and 1.

Proof \Leftarrow :

if $n = ap$ where p is prime, then we show that in the polynomial

$$(x + 1)^{ap} = x^{ap} + \dots + \binom{ap}{p}x^{ap-p} + \dots + 1,$$

the coefficient $\binom{ap}{p}$ is not 0 (mod $n = ap$). Indeed,

$$\binom{ap}{p} = \frac{ap(ap-1)\dots(ap-p+1)}{p!} = \frac{a(ap-1)\dots(ap-p+1)}{(p-1)!}.$$

It is easy to show that $p \nmid (ap-1)\dots(ap-p+1)$. Hence we have shown that $ap \nmid \binom{ap}{p}$.

This means that we only have to test whether

$$(x + 1)^n \equiv x^n + 1 \pmod{n}.$$

This seems trivial, however our lemma leans on picking numbers from a domain greater than n , however the degree of these polynomials is also n .

Define $p(x) = (x + 1)^n - x^n - 1$.

Take a random polynomial $q(x)$ of degree $O(\log n)$ with coefficients from \mathbb{Z}_n .

Check if $p(x) = 0 \pmod{q(x), n}$.

Note: This means that, when we divide by $q(x)$, the arithmetic is done mod n .

Claim: if $p(x) \not\equiv 0$, then

$$\Pr_q [q \nmid p] \geq \frac{1}{\text{degree}(q)} = \frac{1}{\log n}$$

.

We should note the analogy between the two factorization problems: For the polynomial q of degree d :

$\Pr_q [q \text{ is irreducible}] \geq \frac{1}{O(d)}$

For the number $1 \leq i \leq n$:

$\Pr_i [i \text{ is a prime number}] \geq \frac{1}{O(\ln n)}$

So,

$$\Pr_q [\text{error}] \leq \left(1 - \frac{1}{\log n}\right).$$

If we repeat the algorithm t times, we get

$$\Pr_q [\text{error}] \leq \left(1 - \frac{1}{\log n}\right)^t \leq e^{-\frac{t}{\log n}}$$

since $1 + x \leq e^x$.

For this analysis of correctness to work, we must check first that n is not a power of any number. This can be easily done efficiently (e.g., by binary search).

3 Complexity Classes

- $L \in \mathbf{P}$ if there exists deterministic polynomial running time Turing-machine deciding L .
- $L \in \mathbf{RP}$ (Random Polynomial Time) if there exists a probabilistic polynomial running time Turing-machine A such that

- $x \in L \Rightarrow \Pr_r [A(x, r) \text{ accepts}] \geq \frac{1}{2}$
- $x \notin L \Rightarrow \Pr_r [A(x, r) \text{ accepts}] = 0$.

It's still an open question whether $\mathbf{RP} = \mathbf{P}$.

- $L \in \mathbf{BPP}$ (Bounded Probability Polynomial Time) if there exists a probabilistic polynomial running time Turing-machine A such that

- $x \in L \Rightarrow \Pr_r [A(x, r) \text{ accepts}] \geq \frac{2}{3}$
- $x \notin L \Rightarrow \Pr_r [A(x, r) \text{ accepts}] \leq \frac{1}{3}$.

So, \forall input x ,

$$\Pr_r [A(x, r) \text{ accepts}] \in \left[0, \frac{1}{3}\right] \cup \left[\frac{2}{3}, 1\right].$$

It's also an open question whether $\mathbf{BPP} = \mathbf{P}$.

We have shown above that Polynomial Identity Testing for arithmetic formulas is in $\mathbf{co-RP}$.

- $L \in \mathbf{LOGSPACE}$ if there exists a deterministic logarithmic space Turing-machine deciding L .
- $L \in \mathbf{RL}$ if there exists probabilistic logarithmic space Turing-machine A such that

- $x \in L \Rightarrow \Pr_r [A(x, r) \text{ accepts}] \geq \frac{1}{2}$
- $x \notin L \Rightarrow \Pr_r [A(x, r) \text{ accepts}] = 0$.

4 Trivial Derandomization

Given a probabilistic algorithm $A(x, r)$ for a problem, we can convert it into a deterministic algorithm as follows. Enumerate all possible r 's. If $A(x, r)$ accepts in more than half of the cases, then accept, otherwise reject.

In general, if $|r| = \text{poly}(|x|)$ then the number of iterations is $2^{\text{poly}(|x|)}$, and so the runtime is $2^{(\text{poly}(|x|))} \text{poly}(n)$. However, if n is small, this algorithm may be efficient. For example, if $r = O(\log n)$, for inputs of size n , we get $2^{O(\log n)} \text{poly}(n) = \text{poly}(n)$ time.

5 PRG - PseudoRandom Generators

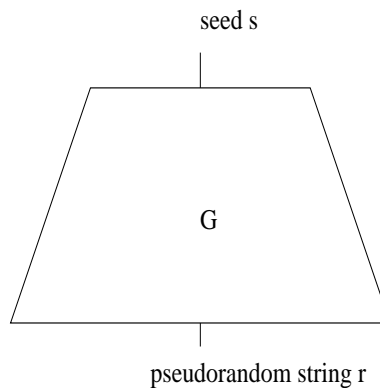


Figure 4: PseudoRandom Generator - model

If we had a machine which generated r pseudorandom bits from s random bits (see Figure 4), then for $|S| = \log |r|$ we could define a new algorithm A' the way Figure 5 describes:

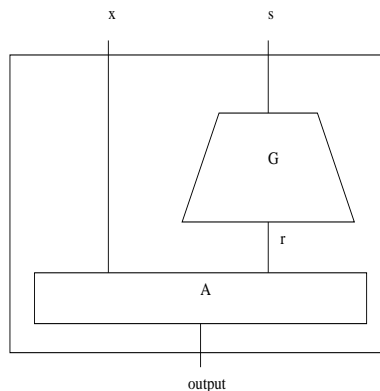


Figure 5: Model of the deterministic algorithm.

We also want to get "fairly good" pseudorandom strings. More formally, we want the PRG G such

that \forall polynomial time algorithm $A(x, r)$, \forall input x ,

$$\Pr_r [A(x, r) = 1] - \Pr_r [A(x, G(s)) = 1] \leq \varepsilon < \frac{1}{6}.$$

This is good because it allows ε to be chosen such that

$$\frac{1}{3} + \varepsilon < \frac{2}{3} - \varepsilon$$

(see definition of the complexity class **BPP**).