CS225: Pseudorandomness Prof. Salil Vadhan

Lecture 2: Randomized Algorithms and Complexity Classes

February 3, 2007

Based on scribe notes by Grant Schoenebeck.

1 Polynomial Identity Testing

Before we study the derandomization of randomized algorithms, we will need some algorithms to derandomize. This section introduces one such algorithm.

The problem is IDENTITY TESTING: given two multivariate polynomials, $p(x_1, \ldots, x_n)$ and $q(x_1, \ldots, x_n)$, decide whether $p \equiv q$ (that is, p is "equivalent" to q).

This definition requires some clarification. Specifically, we need to say what we mean by:

- "polynomials": A (multivariate) polynomial is an finite sum of the form $p(x_1, \ldots, x_n) = \sum_{i_1,\ldots,i_n} c_{i_1,\ldots,i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. We need to specify what space the coefficients of the polynomials will come from; they could be the integers, reals, rationals, etc. In general, we will assume that the coefficients are chosen from a *field* (a set with addition and multiplication, where every nonzero element has a multiplicative inverse) or more generally an *(integral) domain* (where the product of two nonzero elements is always nonzero). Examples of fields include $\mathbb{Q}$ (the rationals), $\mathbb{R}$ (the reals), $\mathbb{Z}_p$ (integers modulo p) where p is prime. An integral domain that is not a field is $\mathbb{Z}$ (the integers); every integral domain is contained in its *field of fractions*, which is $\mathbb{Q}$ in the case of $\mathbb{Z}$. $\mathbb{Z}_n$ for composite n is not even an integral domain. We remark that there does exist a finite field $\mathbb{F}_q$ of size $q = p^k$ for every prime p and positive integer k, and in fact this field is unique (up to isomorphism); but $\mathbb{F}_q$ is only equal to $\mathbb{Z}_q$ in case q is prime (i.e. $k = 1$).

  For a polynomial $p(x_1, \ldots, x_n) = \sum_{i_1,\ldots,i_n} c_{i_1,\ldots,i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$, we define its *degree* to be the maximum of the sum of the exponents $i_1 + \cdots + i_n$ over its monomials with nonzero coefficients $c_{i_1,\ldots,i_n}$. Its *degree in* x_j is the maximum of i_j over its monomials with nonzero coefficients.

- "$p \equiv q$": What could it mean for two polynomials to be equivalent? There are two natural choices: the polynomials are the same as *functions* (they have the same output for every point in the domain), or the polynomials are the same as *formal polynomials* (the coefficients for each monomial are the same).

  Notice that although these two definitions coincide when the polynomials have integral coefficients (or more generally over infinite domains), they do *not* over finite fields. For example, consider

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

  It is easy to see that $p(\alpha) \neq 0$ for all $\alpha \in \mathbb{F}$, but $p \not\equiv 0$ as a formal polynomial.

  For us, equivalence refers to equivalence as formal polynomials.

- "given": What does it mean to be given two polynomials? There are several possibilities here:

  1. Lists of coefficients: this trivializes the problem.
  2. As "oracles": black boxes that, given any point in the domain, gives the value of the polynomial.
  3. As an arithmetic formula: a sequence of symbols like $(x_1 + x_2)(x_3 + x_7 + 6x_5)x_3(x_5 - x_6) + x_2x_4(2x_3 + 3x_5)$ that describes the polynomial. Why can't we solve the problem by just expanding the polynomials?

     More general than formulas are *circuits*. An arithmetic circuit consists of a directed acyclic graph, where internal nodes are labelled by operations ($+$ or $\times$) specifying how to compute a value given the values at its children. Thus we can specify a computation from the input variables (as well as constants in the field) to a special output node; every such circuit defines a polynomial in its input variables $x_1, \ldots, x_n$.

The randomized algorithm that we will use will actually solve a different but equivalent problem.

IDENTITY TESTING (reformulation): Given a polynomial $p(x_1, \ldots, x_n)$, is $p \equiv 0$?

Algorithm for IDENTITY TESTING: Given $p(x_1, \ldots, x_n)$ over field $\mathbb{F}$ of degree at most d,

1. Let $S \subseteq \mathbb{F}$ be any set of size $2d$.
2. Choose $\alpha_1, \ldots, \alpha_n \stackrel{R}{\leftarrow} S$.
3. Evaluate $p(\alpha_1, \ldots, \alpha_n)$. If the result is 0, accept. Otherwise, reject.

It is clear that if $p \equiv 0$, the algorithm will always accept. The correctness in case $p \not\equiv 0$ is based on the following simple but very useful lemma.

Lemma 1 (Schwartz-Zippel) *If p is a nonzero polynomial of degree d over a a field (or integral domain) $\mathbb{F}$ and $S \subseteq \mathbb{F}$, then*

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

In the univariate case ($n = 1$), this reduces to the familiar fact that a polynomial with coefficients in a field and degree d has at most d roots. The proof for multivariate polynomials proceeds by induction on n, and we omit it. (It is a good exercise, and you can also find it in Sudan's algebra notes referenced on the syllabus.)

By the Schwartz-Zippel lemma, the algorithm will err with probability at most $1/2$ when $p \not\equiv 0$. This error probability can be reduced by repeating the algorithm many times (or by increasing $|S|$). Note that the error probability is only over the coin tosses of the algorithm, not over the input polynomial p. This is what we mean when we say *randomized algorithm*; it should work on a worst-case input with high probability over the coin tosses of the algorithm. Algorithms whose

correctness (or efficiency) only holds for randomly chosen inputs are called *heuristics*, and their study is called *average-case analysis*.

Note that we need a few things to ensure that our algorithm will work.

- The first thing that we need is a bound on the degree of the polynomial. We can get this in different ways depending on how the polynomial is represented. For example, in the formula representation (mentioned last time), the degree is bounded by the length of the formula. For circuits, the degree is at most exponential in the size (or even depth) of the circuit.
- We also must be able to evaluate p when the variables take arbitrary values in some set S of size $2d$. For example, if we are given $p : \mathbb{F}^n \to \mathbb{F}$ as an oracle, then we can do this provided that $|\mathbb{F}| \geq 2d$, and we are given an explicit representation of the field $\mathbb{F}$ (e.g. the prime p in case $\mathbb{F} = \mathbb{Z}_p$).

Since these two conditions are satisfied, we have a polynomial-time randomized algorithm for Identity Testing for polynomials given as arithmetic formulas over $\mathbb{Z}$. It can be generalized to circuits (what is the subtlety?). There are no known subexponential-time deterministic algorithms for this problem, even for formulas in $\Sigma\Pi\Sigma$ form. A deterministic polynomial-time algorithm for $\Sigma\Pi\Sigma$ formulas where the outermost sum has only a constant number of terms was given only in 2005.

1.1 Application to Perfect Matching

Definition 2 *Let $G = (V, E)$, then a* matching *on G is a set $E' \subset E$ such that no two edges in E' have a common endpoint.*

Definition 3 *A* perfect matching *on $G = (V, E)$ is a matching such that every vertex is incident to an edge in the matching.*

An important algorithmic problem is Perfect Matching: given a bipartite graph G (with n vertices in each partition), decide whether there a perfect matching in G.

Perfect Matching has a polynomial-time algorithm, as seen in AM 107 (using alternating paths) or CS 226r (by reduction to Max Flow). However, both of these algorithms seem to be inherently sequential in nature. With randomization, we can obtain an efficient parallel algorithm.

Algorithm for Perfect Matching: On input a bipartite graph G with n vertices on each side, we construct an $n \times n$ matrix where

$$A_{i,j}(x) = \begin{cases} x_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Then

$$\det(A(x)) = \sum_{\sigma \in S_n} (-1)^{\text{sign}(\sigma)} \prod_i A_{i,\sigma(i)}$$

but the σ'th term is nonzero if and only if the σ defines a perfect matching. That is, $(i, \sigma(i)) \in E$ for all $1 \leq i \leq n$. So $\det(A(x)) \equiv 0$ iff G has no perfect matching. Moreover its degree is

bounded by n, and given values $\alpha_{i,j}$ for the $x_{i,j}$'s we can evaluate $\det(A(\alpha))$ efficiently in parallel (in polylogarithmic time using a polynomial number of processors) using an efficient parallel algorithm for determinant.

So to test for a perfect matching efficiently in parallel, just run the IDENTITY TESTING algorithm with, say, $S = \{1, \ldots, 2n\} \subset \mathbb{Z}$, to test whether $\det(A(x)) \equiv 0$.

Some remarks:

- A more sophisticated version of this algorithm (choosing the $\alpha_{i,j}$'s from a carefully chosen set S) actually enables finding a perfect matching efficiently in parallel.
- IDENTITY TESTING has been also used to obtain a randomized algorithm for PRIMALITY, which was later derandomized to obtain the celebrated deterministic polynomial-time algorithm for PRIMALITY. You may see this on PS2.

2 Model of Randomized Computation

To develop a rigorous theory of randomized algorithms, we need to use a precise model of computation. There are several possible ways to augmenting a standard deterministic computational model (e.g. Turing machine or RAM model), such as:

1. The algorithm has access to a "black box" that provides it with (unbiased and independent) random bits on request, with each request taking one time step. This is the model we will use.
2. The algorithm has access to a black box that, given a number n in binary, returns a number chosen uniformly at random from $\{1, \ldots, n\}$. This model is often more convenient for describing algorithms. On Problem set 1, you will show that it is equivalent to the Model 1, in the sense that any problem that can be solved in polynomial time on one model can also be solved in polynomial time on the other.
3. The algorithm is provided with an infinite tape (i.e. sequence of memory locations) that is that is initially filled with random bits. For polynomial-time algorithms, this is equivalent to the Model 1. (Why?) However, for space-bounded algorithms, this model seems stronger, as it provides the algorithm with free storage of its random bits (i.e. not counted towards its working memory). Model 1 is considered to be the "right" model for space-bounded algorithms.

3 Complexity Classes

We will now define complexity classes that capture the power of efficient randomized algorithms. As is common in complexity theory, these classes are defined in terms of decision problems, where the set of inputs where the answer should be 'yes' is specified by a *language* $L \subseteq \{0,1\}^*$.

Recall that we say an algorithm A runs in time $t : \mathbb{N} \to \mathbb{N}$ if A takes at most $t(|x|)$ steps on every input x, and it runs in *polynomial time* if it runs time $t(n) = O(n^c)$ for some constant c. Polynomial

time is a theoretical approximation to feasible computation, with the advantage that it is robust to reasonable changes in the model of computation and representation of the inputs.

Definition 4 **P** *is the class of languages L for which there exists a polynomial-time algorithm A such that*

- $x \in L \Rightarrow A(x)$ *accepts.*
- $x \notin L \Rightarrow A(x)$ *rejects.*

For a randomized algorithm A, we say that A runs in time $t : \mathbb{N} \to \mathbb{N}$ if A takes at most $t(|x|)$ steps on every input x *and every sequence of random coin tosses.*

Definition 5 **RP** *is the class of languages L for which there exists a probabilistic polynomial-time algorithm A such that*

- $x \in L \Rightarrow \Pr[A(x)\ \textit{accepts}] \geq \frac{1}{2}$
- $x \notin L \Rightarrow \Pr[A(x)\ \textit{accepts}] = 0$

Thus **RP** captures efficient randomized computation with *one-sided error*. **RP** stands for "random polynomial time". Note that the error probability of an **RP** algorithm can be reduced to $2^{-p(n)}$ for any polynomial p by running the algorithm $p(n)$ times independently and accepting the input iff at least one of the trials accepts. By the same reasoning, the $\frac{1}{2}$ in the definition is arbitrary, and any constant $\alpha \in (0, 1)$ of every $\alpha = 1/\text{poly}(n)$ would yield the same class of languages.

A central question in this course is whether randomization enables us to solve more problems in polynomial time (e.g. decide more languages):

Open Problem 6 *Does* $\mathbf{P} = \mathbf{RP}$*?*

Definition 7 **co-RP** *is the class of languages L whose complement $\bar{L}$ is in* **RP**. *Equivalently, $L \in$* **co-RP** *if there exists a probabilistic polynomial-time algorithm A such that*

- $x \in L \Rightarrow \Pr[A(x)\ \textit{accepts}] = 1$
- $x \notin L \Rightarrow \Pr[A(x)\ \textit{accepts}] \leq \frac{1}{2}$

That is, in **co-RP** we may err on NO instances, whereas in **RP** we may err on YES instances.

Using the IDENTITY TESTING algorithm we saw earlier, we have:

Theorem 8 *The language*

$$\text{ACIT}_{\mathbb{Z}} = \{C : C(x_1, \ldots, x_n) \textit{ an arithmetic circuit over } \mathbb{Z} \textit{ s.t. } C \equiv 0\}$$

is in **co-RP**.

On Problem Set 1, you will show that PRIMALITY is also in **co-RP**. In fact the algorithm you will give is the same one that was later derandomized in the breakthrough result that PRIMALITY is in **P** (2002).

It is common to also allow two-sided error in randomized algorithms:

Definition 9 **BPP** *is the class of languages L such that there exists a probabilistic polynomial-time algorithm A such that*

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

Just as with **RP**, the error probability of **BPP** algorithms can be reduced from 1/3 (or even $1/2 - 1/\text{poly}(n)$) to exponentially small by repetitions, this time taking a majority vote of the outcomes. Proving this requires some facts from probability theory, which we will review shortly.

The cumbersome notation **BPP** stands for 'bounded-error probabilistic polynomial-time," due to the unfortunate fact that **PP** ("probabilistic polynomial-time") refers to the definition where the inputs in the language are accepted with probability greater than 1/2 and inputs not in the length are accepted with probability at most 1/2. Despite its name, **PP** is not a reasonable model for randomized algorithms, as it takes exponentially many repetitions to reduce the error probability. **BPP** is considered the standard complexity class associated with probabilistic polynomial-time algorithms, and thus the main question of this course is:

Open Problem 10 *Does* $\mathbf{BPP} = \mathbf{P}$*?*

So far, we have considered randomized algorithms that can output an incorrect answer if they are unlucky in their coin tosses; these are called 'Monte Carlo' algorithms. It is sometimes preferable to have 'Las Vegas' algorithms, which always output the correct answer, but may run for a longer time if they are unlucky in their coin tosses. For this, we say that A has *expected running time* $t : \mathbb{N} \to \mathbb{N}$ if for every input x, the expectation of the number of steps taken by $A(x)$ is at most $t(|x|)$, where the expectation is taken over the coin tosses of A.

Definition 11 **ZPP** *is the class of languages L such that there exists a probabilistic algorithm A that always decides L correctly and runs in expected polynomial time.*

ZPP stands for "zero-error probabilistic polynomial time". The following relation between **ZPP** and **RP** is left as an exercise.

Fact 12 $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{co\text{-}RP}$.

We do not know any other relations between the classes associated with probabilistic polynomial time.

Open Problem 13 *Are any of the inclusions* $\mathbf{P} \subset \mathbf{ZPP} \subset \mathbf{RP} \subset \mathbf{BPP}$ *proper?*

One can similarly define randomized complexity classes associated with other complexity measures, not just time, such as space or parallel computation. For example:

Definition 14 RNC *is the class of languages L such that exists a probabilistic parallel algorithm A that runs in polylogarithmic time on polynomially many processors, such that*

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

A formal model of a parallel algorithm is beyond the scope of this course, but can be found in standard texts on algorithms or parallel computation. We have seen:

Theorem 15 Perfect Matching, *i.e. the language* $\mathrm{PM} = \{G : G \text{ a bipartite graph with a perfect matching}\}$ *is in* **RNC**.

4 Tail Inequalities

In the previous section, we claimed that we can reduce the error of a **BPP** algorithm by taking independent repetitions and ruling by majority vote. The intuition that this should work is based on the Law of Large Numbers: if we repeat the algorithm many times, the fraction of correct answers should approach its expectation, which is greater than 1/2 (and thus majority rule will be correct). For complexity purposes, we need quantitative forms of this fact, which bound how many repetitions are needed to achieve a desired probability of correctness.

First, we recall a basic inequality which says that it is unlikely for (a single instantiation of) a random variable to exceed its expectation by a large factor.

Lemma 16 (Markov's Inequality) *If X is a nonnegative random variable, then for any $t > 0$,*

$$\Pr[X \geq t] \leq \frac{\mathrm{E}[X]}{t}$$

Markov's Inequality alone does not give very tight concentration around the expectation; to get even a 50% probability, we need to look at deviations by a factor of 2. To get tight concentration, we need to take independent copies of a random variable. There are a variety of different tail inequalities that apply for this setting; they are collectively referred to as *Chernoff Bounds*.

Theorem 17 (A Chernoff Bound) *Let $X_1, \ldots, X_t$ be independent random variables taking values in the interval $[0, 1]$, let $X = (\sum_i X_i)/t$, and $\mu = \mathrm{E}[X]$. Then*

$$\Pr[|X - \mu| \geq \varepsilon] \leq 2\exp(-t\varepsilon^2/2).$$

Thus, the probability of the average deviating from the expectation vanishes exponentially with the number of repetitions t. You will prove this Chernoff Bound on Problem Set 1.

Now let's apply the Chernoff Bound to analyze error-reduction for **BPP** algorithms.

Proposition 18 *The following are equivalent:*

1. $L \in \mathbf{BPP}$.
2. *For every polynomial p, L has a probabilistic polynomial-time algorithm with two-sided error at most $2^{-p(n)}$.*
3. *There exists a polynomial q such that L has a probabilistic polynomial-time algorithm with two-sided error at most $1/2 - 1/q(n)$.*

Proof: Clearly, (2)$\Rightarrow$ (1) $\Rightarrow$ (3). Thus, we prove (3) $\Rightarrow$ (2).

Given an algorithm A with error probability at most $1/2 - 1/q(n)$, consider an algorithm A', which on an input x of length n, runs A for $t(n)$ independent repetitions and rules according to the majority, where $t(n)$ is a polynomial to be determined later.

We now compute the error probability of A' on an input x of length n. Let X_i be an indicator random variable that is 1 iff the i'th execution of $A(x)$ outputs the correct answer, and let $X = (\sum_i X_i)/t$ be the average of these indicators, where $t = t(n)$. Note that $A'(x)$ is correct when $X > 1/2$. By the error probability of A and linearity of expectations, we have $\mathrm{E}[X] \geq 1/2 + 1/q$, where $q = q(n)$. Thus, applying the Chernoff Bound with $\varepsilon = 1/q$, we have:

$$\Pr[X \leq 1/2] \leq 2 \cdot e^{-t/2q^2} < 2^{-p(n)},$$

for $t(n) = 2p(n)q(n)^2$ and sufficiently large n. ■