CS225: Pseudorandomness Prof. Salil Vadhan

Lecture 5: Basic Derandomization Techniques

February 15, 2007

Based on scribe notes by Arthur Rudolph and Chun-Yun Hsiao.

1 Recap

Over the past few lectures, we have discussed some striking examples of the power of randomness for the design of efficient algorithms:

- Identity Testing in **co-RP**.
- ε-Approx #DNF in **prBPP**.
- Perfect Matching in **RNC**.
- Undirected S-T Connectivity in **RL**.
- Large Cut in probabilistic polynomial time.

This is of course only a small sample; there are entire courses on ways of exploiting randomness in computation (e.g. CS223, CS 224r, MIT 6.856). One topic in particular we omitted is the usefulness of randomness for *verifying proofs*. Recall that **NP** is the class of language having membership proofs that can be verified in **P**. Thus it is natural to consider proof verification that is probabilistic, leading to the class **MA**, as well as a larger class **AM**, where the proof itself can depend on the randomness chosen by the verifier. (These are both subclasses of the class **IP** of languages having *interactive proof systems.*) There are languages, such as Graph Nonisomorphism, that are in **AM** but are not known to be in **NP**. "Derandomizing" these proof systems (e.g. proving **AM** = **NP**) would mean showing that Graph Nonisomorphism is in **NP**, i.e. there are short proofs that two graphs are nonisomorphic. You can read more about interactive proofs in the lecture notes from Spring 2004.

In the rest of the course, we will turn towards *derandomization* — trying to remove the randomness from these algorithms. We will achieve this for some of the specific algorithms we studied, and also attack the larger questions of whether all efficient randomized algorithms can be derandomized, e.g. does **BPP** = **P**? **RL** = **L**?, **RNC** = **NC**?

Over the next couple of lectures, we will introduce a variety of "basic" derandomization techniques. These will be deficient in that they either are infeasible (e.g. cannot be carried in polynomial time) or very specialized (e.g. apply only in very specific circumstances). But it will be useful to have these as tools before we proceed to study more sophisticated pseudorandom objects.

2 Enumeration

We are interested in how much savings randomization provides. One way of asking this is to try to find the smallest possible upper bound on the deterministic time complexity of languages in **BPP**.

Definition 1

$$\begin{aligned}
\mathbf{DTIME}(t(n)) &= \{L : L \textit{ can be decided deterministically in time } O(t(n))\} \\
\mathbf{P} &= \cup_c \mathbf{DTIME}(n^c) \\
\tilde{\mathbf{P}} &= \cup_c \mathbf{DTIME}(2^{(\log n)^c}) \\
\mathbf{SUBEXP} &= \cap_\varepsilon \mathbf{DTIME}(2^{n^\varepsilon}) \\
\mathbf{EXP} &= \cup_c \mathbf{DTIME}(2^{n^c})
\end{aligned}$$

The "Time Hierarchy Theorem" (found in any complexity text, e.g. Sipser) implies that all of these classes are distinct, i.e. $\mathbf{P} \subsetneq \tilde{\mathbf{P}} \subsetneq \mathbf{SUBEXP} \subsetneq \mathbf{EXP}$. More generally, it says that $\mathbf{DTIME}(o(t(n)/\log t(n))) \subsetneq \mathbf{DTIME}(t(n))$ for any efficiently computable time bound t. (What is difficult in complexity theory is separating classes that involve different computational resources, like determinstic time vs. nondeterministic time.)

Enumeration enables us to deterministically simulate any randomized algorithm with an exponential slowdown.

Proposition 2 $\mathbf{BPP} \subseteq \mathbf{EXP}$.

Proof: If L is in **BPP**, then there is a probabilistic polynomial-time algorithm for A running in time $t(n)$ for some polynomial t. As an upper bound, A uses at most $t(n)$ random bits. We can view A as a deterministic algorithm on two inputs — its regular input x and its coin tosses r. We'll write $A(x;r)$ for A's output.

$$\Pr[A(x;r) \text{ accepts}] = \frac{1}{2^{t(n)}} \sum_{r \in \{0,1\}^{t(n)}} A(x;r)$$

We can compute the right-hand side of that expression in deterministic time $2^{t(n)} \cdot t(n)$. ■

We see that the enumeration method is *general* in that it applies to all **BPP** algorithms, but it is *infeasible* (taking exponential time). However, if the algorithm only uses a small number of random bits, it becomes feasible:

Proposition 3 *If L has a probabilistic polynomial-time algorithm that runs in time $t(n)$ and uses $r(n)$ random bits, then $L \in \mathbf{DTIME}(t(n) \cdot 2^{r(n)})$. In particular, if $t(n) = \mathrm{poly}(n)$ and $r(n) = O(\log n)$, then $L \in \mathbf{P}$.*

Open Problem 4 *Is* **BPP** *closer to* **P** *or* **EXP**? *Is* $\mathbf{BPP} \subseteq \tilde{\mathbf{P}}$? *Is* $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$?

3 Nonconstructive/Nonuniform Derandomization

Proposition 5 *If $A(x;r)$ is a randomized algorithm for a language L that has error probability smaller than 2^{-n} on inputs x of length n, then for every n, there exists a fixed sequence of coin tosses r_n such that $A(x;r_n)$ is correct for all $x \in \{0,1\}^n$.*

Proof: Consider R_n chosen uniformly at random. Then

$$\begin{aligned} &\Pr[\exists x \in \{0,1\}^n \ A(x;R_n) \text{ incorrect on } x] \\ \leq \ & \sum_x \Pr[A(x;R_n) \text{ incorrect on } x] \\ < \ & 2^n \cdot 2^{-n} < 1 \end{aligned}$$

Thus, there exists a fixed value $R_n = r_n$ that is correct for all $x \in \{0,1\}^n$. ■

The advantage of this method over enumeration is that once we have the fixed string r_n, computing $A(x;r_n)$ can be done in polynomial time. However, the proof that r_n exists is nonconstructive; it is not clear how to find it in less than exponential time.

Note that we know that we can reduce the error probability of any **BPP** (or **RP**, **RL**, **RNC**, etc.) algorithm to smaller than 2^{-n} by repetitions, so we can apply this proposition in general. However, we begin by looking at some interesting special cases.

Example (Perfect Matching). Let $G = (V,E)$ be a bipartite graph with m vertices on each side, and let $A(x_{1,1},\ldots,x_{m,m})$ be the matrix that $A_{i,j} = x_{i,j}$ if $(i,j) \in E$, and $A_{i,j} = 0$ if $(i,j) \notin E$. Recall from Lecture 2 that the polynomial $\det(A(x))$ is nonzero if and only if G has a perfect matching. Let $S = \{0,1,2,\ldots,m2^{m^2}\}$. We argued that if we choose $\alpha \overset{\text{R}}{\leftarrow} S^{m^2}$ at random and evaluate $\det(A(\alpha))$, we can determine whether $\det(A(x))$ is zero with error probability at most $m/|S|$ which is smaller than 2^{-m^2}. Since a bipartite graph on m vertices is specified by a string of length $n = m^2$, by Proposition 5 we know that there exists an α such that $\det(A(\alpha)) \neq 0$ if and only if G has a perfect matching.

Open Problem 6 *Can we find such an $\alpha = (\alpha_{1,1},\ldots,\alpha_{m,m})$ explicitly, i.e.,* deterministically *and* efficiently*? By efficiently, we mean polynomial time (in m), though an* **NC** *algorithm would be preferable (for placing* Perfect Matching *in (uniform)* **NC***), while even a subexponential-time algorithm would be interesting.*

Example (Universal Traversal Sequences). Let G be a connected d-regular unidrected multigraph of n vertices. From the previous lecture, we know that a random walk of $\text{poly}(n,d)$ steps from any start vertex will visit any other vertex with high probability. By increasing the length of the walk by a polynomial factor, we can ensure that every vertex is visited with probability greater than $1 - 2^{-nd\log n}$. By the same reasoning in last example, we know that for every pair of (n,d), there exists a *universal traversal sequence* $w \in \{1,2,\ldots,d\}^{\text{poly}(n,d)}$ such that for any such G and any vertex s in G, if we start from s and follow w we will cover the entire graph.

Open Problem 7 *Can we construct such a universal traversal sequence explicitly (e.g. in polynomial time or even logarithmic space)?*

The best known explicit construction of a universal traversal sequence has length (and time) $n^{O(\log n)}$. The methods underlying the recent deterministic logspace algorithm for UNDIRECTED S-T CONNECTIVITY (which we will cover) also yield universal traversal sequences that work on graphs where the labelling of edges satisfy a certain 'consistency' condition. Removing this consistency condition seems to be the main obstacle towards proving $\mathbf{RL} = \mathbf{L}$ in general using those methods.

We now cast the nonconstructive derandomizations provided by Proposition 5 in the language of 'nonuniform' complexity classes.

Definition 8 *Let* $\mathbf{C}$ *be a class of languages, and* $a : \mathbb{N} \to \mathbb{N}$ *be a function. Then* $\mathbf{C/a}$ *is a class of languages defined as follows:* $L \in \mathbf{C/a}$ *if there exists* $L' \in \mathbf{C}$*, and* $\alpha_1, \alpha_2, \ldots \in \{0,1\}^*$ *with* $|\alpha_n| \le a(n)$*, such that* $x \in L \Leftrightarrow (x, \alpha_{|x|}) \in L'$*. The* α*'s are called the* advice *strings.*

One of the most natural nonuniform class is $\mathbf{P/poly} \stackrel{\text{def}}{=} \bigcup_c \mathbf{P}/n^c$. That is, polynomial time with polynomial advice. A basic result in complexity theory is that $\mathbf{P/poly}$ is exactly the class of languages that can be decided by polynomial-sized Boolean circuits. (A language L is decided by a family of polynomial-sized Boolean circuits $\{C_n\}_{n\in\mathbb{N}}$, where $|C_n| \le p(n)$ for some polynomial p, if for all n, $C_n : \{0,1\}^n \to \{0,1\}$ decides $L \cap \{0,1\}^n$.) Although $\mathbf{P/poly}$ contains undecidable problems,[1] people generally believe that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, and indeed trying to prove lower bounds on circuit size is one of the main approaches to proving $\mathbf{P} \neq \mathbf{NP}$, since circuits seem much more concrete and combinatorial than Turing machines. (However this has turned out to be quite difficult; the best circuit lower bound known for computing an explicit function is roughly 4.5n.)

Proposition 5 directly implies:

Corollary 9 $\mathbf{BPP} \subset \mathbf{P/poly}$.

A more general meta-theorem is that "nonuniformity is more powerful than randomness."

4 Nondeterminism

Although somewhat unrealistic, nondeterminism is considered as a powerful resource in complexity theory. Assuming we have this resource, can we guess a "good" random string and then do the computation deterministically? How do we check if a string is good or not? For classes with one-sided error, nondeterminism *is* more powerful than randomness.

Proposition 10 $\mathbf{RP} \subseteq \mathbf{NP}$.

[1]Consider the unary version of halting problem, the advice string α_n is simply a bit that tells us whether the n'th Turing machine halts or not.

Proof: ■

In contrast, we do not know whether **BPP** is contained in **NP**. Indeed, it is consistent with current knowledge that **BPP** = **NEXP**! Nevertheless, assuming **P** = **NP** (in some sense, assuming nondeterminism is "cheap"), we can show that **P** = **BPP**.

Theorem 11 *If* **P** = **NP***, then* **P** = **BPP***.*

Proof Idea: Let $L \in \mathbf{BPP}$, express membership in L using two quantifiers. That is, for some polynomial-time predicate P,

$$x \in L \iff \exists y \forall z\, P(x,y,z)$$

Assuming $\mathbf{P} = \mathbf{NP}$, we can replace $\forall z\, P(x,y,z)$ by a polynomial-time predicate $Q(x,y)$, because $\{(x,y) : \forall z\, P(x,y,z)\} \in \textbf{co-NP} = \mathbf{P}$. Then $L = \{x : \exists y\, Q(x,y)\} \in \mathbf{NP} = \mathbf{P}$.

Fix a randomized algorithm A for a **BPP** language L, and assume, w.l.o.g., that its error probability is smaller than 2^{-n} and it uses $m = \mathrm{poly}(n)$ coin tosses. Let $Z_x \subset \{0,1\}^m$ be the set of coin tosses r for which $A(x;r) = 0$. We will show that if x is in L, there exist m points in $\{0,1\}^m$ such that no "translation" of Z_x covers all the points. Intuitively, this should be possible because Z_x is an exponentially small fraction of $\{0,1\}^m$. On the other hand if $x \notin L$, then for any m points in $\{0,1\}^m$, we will show that there is a "translation" of Z_x that covers all the points. Intuitively, this should be possible because Z_x covers all but an exponentially small fraction of $\{0,1\}^m$.

Proof: Let $s \in \{0,1\}^m$, and define "translation of Z_x" as $Z_x \oplus s = \{b \oplus s : b \in Z_x\}$. We will show

$$\begin{aligned} x \in L \;&\Rightarrow\; \exists r_1, r_2, \ldots, r_m \in \{0,1\}^m \forall s \in \{0,1\}^m \quad \neg \bigwedge_{i=1}^{m} (r_i \in Z_x \oplus s) \\ &\Leftrightarrow\; \exists r_1, r_2, \ldots, r_m \in \{0,1\}^m \,\forall s \in \{0,1\}^m \quad \neg \bigwedge_{i=1}^{m} (A(x; r_i \oplus s) = 0)\,; \end{aligned}$$

$$\begin{aligned} x \notin L \;&\Rightarrow\; \forall r_1, r_2, \ldots, r_m \in \{0,1\}^m \,\exists s \in \{0,1\}^m \quad \bigwedge_{i=1}^{m} (r_i \in Z_x \oplus s) \\ &\Leftrightarrow\; \forall r_1, r_2, \ldots, r_m \in \{0,1\}^m \,\exists s \in \{0,1\}^m \quad \bigwedge_{i=1}^{m} (A(x; r_i \oplus s) = 0)\,. \end{aligned}$$

We prove both parts by the Probabilistic Method.

$x \in L$: Choose $R_1, R_2, \ldots, R_m \overset{\mathrm{R}}{\leftarrow} \{0,1\}^m$. Then, for every fixed s, we have

$$\Pr[R_i \in Z_x \oplus s] < 2^{-n} \quad \Rightarrow \quad \Pr\left[\bigwedge_i (R_i \in Z_x \oplus s)\right] < 2^{-nm}.$$

So

$$\Pr\left[\exists s \bigwedge_i (R_i \in Z_x \oplus s)\right] < 2^m \cdot 2^{-nm} < 1.$$

Thus there exist $r_1, \ldots, r_m$ such that $\forall s \neg \bigwedge_i (r_i \in Z_x \oplus s)$, as desired.

$x \notin L$: Let $r_1, r_2, \ldots, r_m$ be arbitrary, and choose $S \overset{\text{R}}{\leftarrow} \{0,1\}^m$ at random. Then, for each i,

$$\Pr[r_i \notin Z_x \oplus S] = \Pr[S \notin Z_x \oplus r_i] < 2^{-n}.$$

So

$$\Pr\left[\bigvee_i (r_i \notin Z_x \oplus S)\right] < m \cdot 2^{-n} < 1.$$

Thus, for every $r_1, \ldots, r_m$, there exists s such that $\bigwedge_i (r_i \in Z_x \oplus s)$, as desired.

■