CS 744: Pseudorandomness Generators

Lecture 6: Basic Derandomization Techniques

January 16, 2015

Our theme is that randomized algorithms are easy to design, and we hope to systematically derandomize such algorithms to get deterministic polynomial-time solutions to problems. For any given problem, such an approach may not yield the optimal algorithm, but we aim only to show that the problem can be solved in P. What we stand to gain is *simplicity* in the design of the algorithm.

We will look at some elementary, general-purpose derandomization techniques. [1] These techniques are too coarse to yield the best practical derandomizations for specific problems - indeed, some of these are not polynomial-time algorithms or even deterministic. However, these techniques, besides being elementary and hence a first step in derandomization, also serve to show the connections between randomized computation and other forms of resource bounds in computation.

1 Enumeration

Suppose we have a polynomial-time randomized algorithm $A(x;r)$ which on input x and a given random sequence of coin-tosses r outputs an answer. Let $m(n)$ be the polynomial in n which upper bounds the running time of A on any input of length n.

Then we can simulate all possible runs of $A(x;)$ by enumerating all the possible random coin tosses of length $m(n)$ - there are at most $2^{m(n)}$ of these, and collecting the outputs.

Now, there may be various acceptance criteria for $x \in L$ - *e.g* $x \in L$ if and only if there is an $\epsilon > 0$ with more than $1/2 + \frac{1}{|x|^\epsilon}$ of the coin tosses, A outputs "yes". If we have all the possible answers of $A(x;)$, then we can compute the probability of acceptance and simulate any acceptance criterion.

The complexity class BPP is defined in terms of one such acceptance criterion.

Definition 1. *A language L is in BPP if there is a randomized polynomial-time algorithm A*

[1] Material for this chapter is taken from Salil Vadhan's monograph, "Pseudorandomness", Chapter 3.

with time bound $m(n)$ such that for some $c > 0$,

$$x \in L \Rightarrow \frac{\left|\{r \in \{0,1\}^{m(|x|)} \mid A(x,r) = \text{“yes”}\}\right|}{2^{m(|x|)}} \geq \frac{1}{2} + \frac{1}{|x|^c}$$

$$x \notin L \Rightarrow \frac{\left|\{r \in \{0,1\}^{m(|x|)} \mid A(x,r) = \text{“no”}\}\right|}{2^{m(|x|)}} \geq \frac{1}{2} + \frac{1}{|x|^c}$$

Recall that EXP is the class of problems solvable in $2^{\text{poly}(n)}$ where n is the length of the input. The argument above essentially proves the following fact.

Theorem 2. *BPP $\subseteq$ EXP.*

This derandomization is impractical since you need exponential time to derandomize a polynomial time probabilistic algorithm, hence the overall approach is not better than brute force for many problems.

2 Nonconstructive derandomization

In the probabilistic method [1], in order to demonstrate that there is an object x with property P, we establish that $P(x)$ is satisfied with positive probability. In finite probability spaces, this is sufficient to show that an x with property P has to exist. Often, we prove that $\neg P(x)$ has probability strictly less than 1.

We now show that there are “uniformly applicable” random sequences which work for *any input* of a particular length n. The proof will use the probabilistic method, and we will not mention how to find this nice random sequence in polynomial-time. However, the mere existence of such nice sequences proves something stronger [2] than Theorem 2.

Theorem 3. *Let $A(x;r)$ be a randomized algorithm for deciding a language L, with the probability of error,* i.e. *$A(x;r) \neq L(x)$, $< 2^{-|x|}$ for any x.*

Then for every length n, there is a random string r_n such that for all strings w of length n, $A(w;r_n) = L(w)$.

The power of the theorem is that instead of different random strings working for different inputs, you have a limited notion of a uniformly good random string - there is one random string which gives the correct answer on all 2^n inputs of length n. For different lengths, however, you may need to use different random strings.

[2] according to the current beliefs of computational complexity theorists

Proof. Let n be fixed. Let r be drawn uniformly from $\{0,1\}^{m(n)}$ where $m(n)$ is the number of random bits used in the worst case by the algorithm A on any input of length n. Then

$$\begin{aligned} Pr[\exists x \in \{0,1\}^n \mid A(x;r) \neq L(x)] &\leq \sum_{x \in \{0,1\}^n} Pr[x \in \{0,1\}^n \mid A(x;r) \neq L(x)] \\ &< 2^n 2^{-n} \quad = 1. \end{aligned}$$

Hence there is an $r \in \{0,1\}^{r(n)}$ such that for every n-long string x, $A(x;r) = L(x)$. □

This has a complexity theoretic consequence, with respect to *circuit families*. We give the following informal definition of circuits and circuit families.

Definition 4. *A* circuit *is a directed acyclic graph where the vertices are logical gates or boolean inputs, and the edges represent the connections between the logical gates. Each logical gate has at most 2 inputs. The* size *of the circuit is the number of gates in the circuit.*

A circuit family $\langle C_n \rangle_{n=0}^{\infty}$ *for deciding a language L is a sequence of circuits such that for every length n, the circuit C_n decides the membership of all n-long strings in L.*

Definition 5. *The class P/Poly is the class of languages L decided by some circuit family $\langle C_n \rangle_{n=0}^{\infty}$ deciding L with a polynomial $p(n)$ which upper bounds size(C_n) for all n.*

By Theorem 3, there is one random string which works on all inputs of length n if an algorithm has probability of error $< 2^{-n}$ on all inputs of length n. Informally, we can "hard-code" this random string into a circuit C_n, where C_n emulates the execution of A on n-long strings. If A is polynomial-time, then it is possible to make C_n polynomial-sized. For a detailed argument, see [2]. This argument justifies the following corollary.

Corollary 6. *$BPP \subseteq P/Poly$.*

References

[1] N. Alon and J. Spencer. *The Probabilistic Method.* Wiley and Sons, 3 edition, 2008.

[2] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.