CS 744: Pseudorandomness Generators

Lecture 7: Basic Derandomization Techniques - II

January 18, 2015

1 Derandomization via Nondeterminism

We will now try to study the relationship between BPP and NP through these basic techniques of derandomization. It is not clear whether NP can solve problems which probabilisitic polynomial-time algorithms with bounded *two-sided* error can. This is a famous open question in computational complexity theory. However, something very close to BPP $\subseteq$ NP is known:

Sipser showed, in complexity-theoretic terms, that BPP $\subseteq \Sigma_2 \cap \Pi_2$. In terms of the polynomial-time hierarchy, this states that BPP is not far more powerful than NP is. Avoiding the definition of the Polynomial-time hierarchy, we can state the above result as follows.

Theorem 1. *If P = NP, then P = BPP.*

According to the current belief in complexity theory, we believe that P $\neq$ NP, and P = BPP. Viewed thus, the above theorem is "believed" to be a vacuous implication.

However, the theorem can be viewed in a positive light — it says that non-determinism is almost as powerful as probabilistic computation. If we ponder this for a moment, we see that it is consistent with what we believe the truth to be: BPP = P $\subsetneq$ NP.

Proof. Let $L \in$ BPP. Without loss of generality, assume that A is a BPP algorithm for L with error probability less than 2^{-n}. We will show that

$$x \in L \quad \Leftrightarrow \exists y \forall z P(x, y, z), \tag{1}$$

where P is some polynomial-time computable predicate. This suffices to establish the result because of the following reason.

A language in L_1 in NP can be characterized as

$$x \in L_1 \quad \Leftrightarrow \exists y Q(x, y),$$

where Q is some polynomial-time computable predicate. Consequently, the complement of L_1 can be characterized as

$$u \in L_1^c \quad \Leftrightarrow \forall v R(u, v),$$

where R is some polynomial time predicate (*e.g.* $\neg Q$).

If P is equal to NP, then the complement of each language in NP is also in NP. Hence, we can replace $\forall z P(x,y,z)$ in equation (1) by a polynomial-time computable predicate $R(x,y)$, and equation (1) can be rewritten as

$$x \in L \quad \Leftrightarrow \exists y P(x,y), \tag{2}$$

and it follows that L $\in$ NP.

We now show that L $\in$ BPP can be written in the form (1). Let Y_x be the set of random sequences of length m that cause the algorithm to output "yes". If $x \in L$, then the probability of Y_x is at least $1-2^{-n}$. The idea is to produce a few "shifts" of the set Y_x which cover all the strings in $\{0,1\}^m$. If $x \notin L$, since the probability of Y_x is less than 2^{-n}, these few shifts of Y_x will cover only a very small part of $\{0,1\}^m$.

These shifts will be produced by selecting m strings $s_1, \ldots, s_m$ from $\{0,1\}^m$ and then XOR-ing Y_x with each of them.[1] Since $s_i \oplus y$ is unique for every $y \in Y_x$, we know that

$$|Y_x| = |Y \oplus s_1| = \cdots = |Y \oplus s_m|.$$

We analyze the following cases.

§1. Suppose $x \in L$. Then we show that

$$\exists s_1, \ldots, s_n \in \{0,1\}^m \forall y \in \{0,1\}^m y \oplus s_1 \in Y_x \vee \ldots y \oplus s_m \in Y_x.$$

This is an existential claim, hence we can use the probabilistic method, as in Theorem 2 of Lecture 6. Select $s_1, \ldots, s_m$ uniformly at random from the set of $\{0,1\}^m$. Now, let r be any random sequence from $\{0,1\}^m$.

$$\begin{aligned} Pr\left[r \in \Sigma^m \mid r \notin \bigcup_{i=1}^{m} Y_x \oplus s_i\right] &= \Pi_{i=1}^m Pr[r \notin Y_x \oplus s_i] \\ &= \Pi_{i=1}^m Pr[s_i \notin Y_x \oplus r] \\ &< (2^{-n})^m \end{aligned}$$

Now,

$$Pr[\exists r \in \{0,1\}^m \mid r \notin \bigcup_{i=1}^{m} Y_x \oplus s_i] \quad \leq 2^m 2^{-mn} = 2^{-n}.$$

Hence there exist $s_1, \ldots, s_m$ such that $\bigcup_{i=1}^m A \oplus s_i$ covers $\{0,1\}^m$.

§2. If $x \notin L$, then we can show that for any $s_1, \ldots, s_m$ from $\{0,1\}^m$, there will be a $y \in \{0,1\}^m$ such that

$$y \notin \cup_{i=1}^m Y_x \oplus s_i.$$

This follows by a union bound since Y_x has probability at most 2^{-n} and the union above cannot have probability greater than $m2^{-n}$. Hence there will be strings in $\{0,1\}^m$ which are excluded from it. $\square$

[1] Algorithmically, for every $y \in Y_x$, compute $y \oplus s_1, \ldots, y \oplus s_m$ and say that $x \in L$ if A says yes with of these m resulting strings as the random input, and say no if A says no on all of them.

2 The Method of Conditional Expectations

This is an elegant probabilistic argument. It leads to polynomial-time algorithms unlike the methods we have seen so far. Unfortunately, however, this does not seem to efficiently derandomize all randomized algorithms. Thus it may not derandomize BPP as a whole, only some specific problems. We will describe the general method, skipping how this yields polynomial-time solutions for specific problems.

(To be completed)