

# CMPT 710 - Complexity Theory: Lecture 18

Valentine Kabanets

November 8, 2007

## 1 BPP and P

It is a big open question whether every language in BPP can be decided in *deterministic* polytime, i.e., whether  $\text{BPP} = \text{P}$ . There is some evidence that this equality indeed holds. The reason is the following result.

**Theorem 1** (Impagliazzo, Wigderson). *If  $\text{DTIME}(2^{O(n)})$  contains a language  $L$  of circuit complexity  $2^{\Omega(n)}$ , then  $\text{BPP} = \text{P}$ .*

The proof of this theorem is too involved to be presented in the “Intro to Complexity” course.

Some remarks about the Impagliazzo-Wigderson theorem. First, note that the complete derandomization of BPP is possible if we can efficiently *deterministically* construct truth tables of Boolean functions of near-maximum circuit complexity (since the maximum circuit complexity of an  $n$ -variable Boolean function is about  $2^n/n$ ). The condition that  $L \in \text{DTIME}(2^{O(n)})$  is equivalent to saying that the truth table of some “hard” Boolean function in  $n$  variables can be produced deterministically in time  $\text{poly}(2^n)$ , polynomial in the size of the truth table. Note the interplay of uniform and nonuniform complexities here: We want to produce efficiently *uniformly* (in time  $\text{poly}(2^n)$ ) truth tables of Boolean functions of high ( $2^{\Omega(n)}$ ) *nonuniform* (i.e., circuit) complexity.

The second comment is that there is a tradeoff between circuit complexity of a language in EXP and the efficiency of derandomization. For example, Babai, Fortnow, Nisan, and Wigderson showed that if some language in EXP needs more than polynomial circuit size, then every language in BPP can be decided deterministically in subexponential time, i.e., time  $\text{DTIME}(2^{n^\epsilon})$ , for any  $\epsilon > 0$ .

Finally, there is an intimate relation between derandomizing BPP and proving circuit lower bounds. It is shown by Impagliazzo and myself that in order to prove  $\text{BPP} = \text{P}$ , one would have to prove superpolynomial circuit lower bounds of some kind. Thus, derandomization and circuit complexity are inextricably connected.

## 2 BPP and PH

It is not known how BPP and NP are related to each other. They both belong to PSPACE, however. (For BPP, we just enumerate all random strings of polysize, counting how many of

them are accepted by our randomized polytime algorithm. This can be done in polyspace.)

Actually, even more is known about BPP.

**Theorem 2** (Sipser).  $\text{BPP} \subseteq \Sigma_2^P \cap \Pi_2^P$

*Proof.* Since  $\text{BPP} = \text{coBPP}$  (check this!), it suffices to prove that  $\text{BPP} \subseteq \Sigma_2^P$ .

We'll follow the proof due to Lautemann. The idea is simple. Assume that a given  $L \in \text{BPP}$  is decided by a probabilistic TM  $M$  with error probability less than  $2^{-n}$ , where  $n$  is the input size. As we saw in the last lecture, we can always reduce the error probability to be that low. Fix an input  $x$  of size  $n$ . Consider the set of  $A$  of random strings  $r$  on which our TM  $M$  accepts  $x$ , i.e.,  $A = \{r \mid M(x, r) \text{ accepts}\}$ . Let  $R$  be the set of all random strings  $r$ , for an input of size  $n$ .

There are two cases. Case I:  $x \in L$ . Then, by our assumption,  $\frac{|A|}{|R|} > 1 - 2^{-n}$ . We will consider *translations* of the set  $A$ : given a binary string  $s$ , where  $|s| = |r|$ , we define the set  $A \oplus s = \{a \oplus s \mid a \in A\}$ , where  $a \oplus s$  means the bitwise XOR of the strings  $a$  and  $s$ . We will argue that, since  $A$  is big, there will be a small number of strings  $s_1, \dots, s_k$ , for  $k = |r|$ , such that

$$R = \cup_{i=1}^k A \oplus s_i,$$

i.e., translating the set  $A$  for  $k$  times will cover the entire set  $R$ .

**Claim 3.** *If  $|A|/|R| > 1 - 2^{-n}$ , then there exist strings  $s_1, \dots, s_k$ , for  $k = |r|$ , such that  $R = \cup_{i=1}^k A \oplus s_i$ .*

*Proof of Claim.* The proof is an easy application of the *Probabilistic Method*. We'll show that a random collection of  $k$  strings will have the required property with non-zero probability, and so a desired collection of  $s_i$ 's certainly exists.

So, let's pick a random sequence  $s_1, \dots, s_k$  uniformly and independently. Let  $S = \cup_{i=1}^k A \oplus s_i$ . For a fixed string  $r \in R$ , we have

$$\Pr[r \notin S] = \prod_{i=1}^k \Pr[r \oplus s_i \notin A] < (2^{-n})^k = 2^{-nk}.$$

Hence, applying the "union bound",

$$\Pr[\exists r \in R \text{ such that } r \notin S] \leq |R|2^{-nk} = 2^k 2^{-nk} \ll 1.$$

It follows that a randomly chosen sequence  $s_1, \dots, s_k$  is good with probability  $1 - 2^{-nk+k} > 0$ , and hence a good sequence exists.  $\square$

Now, in case II:  $x \notin L$ . We have  $|A|/|R| < 2^{-n}$ . We claim that there is *no* sequence  $s_1, \dots, s_k$  such that  $R = \cup_{i=1}^k A \oplus s_i$  in this case. The proof is very simple:  $|\cup_{i=1}^k A \oplus s_i| \leq k|A| < \frac{k}{2^n}|R| \ll |R|$ . So, we'll never be able to cover the set  $R$  by few "translated" copies of the small set  $A$ .

To summarize, what we proved above is the following:  $x \in L$  iff there exist  $s_1, \dots, s_k$  such that for every  $r \in R$  at least one of  $M(x, r \oplus s_i)$  accepts. But this is exactly a  $\Sigma_2^P$  formula. Hence,  $L \in \Sigma_2^P$ .  $\square$

## 3 Example of a BPP language

### 3.1 Polynomial Identity Testing

Suppose we are given two arithmetic formulas  $f(x_1, \dots, x_n)$  and  $g(x_1, \dots, x_n)$  with integer coefficients. We want to know whether these formulas are equivalent, i.e., whether  $f \equiv g$  as polynomials. One way to check this is to write out all the coefficients of the polynomials  $f$  and  $g$ , and compare them one by one. However, there may be exponentially many such coefficients, and so this approach will result in a highly inefficient algorithm.

A better way to solve this problem is by using a randomized algorithm. We simply pick random values to the variables  $x_1, \dots, x_n$ , and check if the two polynomials agree on these values. If the two polynomials are equivalent, then they will evaluate to the same value on any given point. If they are different polynomials, then it's very unlikely that they will agree on a random point. To formalize this, we need the following lemma. Recall that the degree of a monomial  $x_1^{d_1} \dots x_n^{d_n}$  is  $d_1 + \dots + d_n$ ; the total degree of a polynomial  $f$  is the maximum degree of a monomial of  $f$ .

**Lemma 4** (Schwartz-Zippel, and many others). *Let  $f(x_1, \dots, x_n)$  be a non-zero polynomial over a field  $F$ , such that the total degree of  $f$  is  $d$ . Let  $S \subseteq F$  be a finite subset of field elements. Then*

$$\Pr_{r_1, \dots, r_n \in S}[f(r_1, \dots, r_n) = 0] \leq \frac{d}{|S|}.$$

*Proof.* By induction on the number of variables  $n$ . Base case of  $n = 1$  is easy: a univariate degree  $d$  non-zero polynomial over a field can have at most  $d$  roots. Hence, a random point  $r$  is a root with probability at most  $d/|S|$ .

Assume the statement is true for  $k$  variables, and let's prove it for  $k+1$  variables. Express a polynomial  $f(x_1, \dots, x_{k+1})$  as a polynomial  $f_1(x_{k+1})$ , whose coefficients are polynomials in  $x_1, \dots, x_k$  (by factoring out the expressions  $x_{k+1}^i$  for  $1 \leq i \leq d$ ). Let  $d_1$  be the degree in  $x_{k+1}$  of  $f_1$ . Let  $p(x_1, \dots, x_k)$  be the coefficient at  $x_{k+1}^{d_1}$  in  $f_1$ ; that is,  $f_1(x_{k+1}) = x_{k+1}^{d_1} p(x_1, \dots, x_k) + \dots$ . Note that the total degree of  $p$  is at most  $d - d_1$ , since  $d$  is the total degree of  $f$ . We have  $\Pr_{r_1, \dots, r_{k+1}}[f(r_1, \dots, r_{k+1}) = 0] =$

$$\Pr[f(r_1, \dots, r_{k+1}) = 0 \mid p(r_1, \dots, r_k) \neq 0] * \Pr[p(r_1, \dots, r_k) \neq 0] \tag{1}$$

$$+ \Pr[f(r_1, \dots, r_{k+1}) = 0 \mid p(r_1, \dots, r_k) = 0] * \Pr[p(r_1, \dots, r_k) = 0]. \tag{2}$$

We can upperbound this sum as follows.

$$\Pr[f(r_1, \dots, r_{k+1}) = 0 \mid p(r_1, \dots, r_k) \neq 0] * \Pr[p(r_1, \dots, r_k) \neq 0] \leq$$

$$\Pr[f(r_1, \dots, r_{k+1}) = 0 \mid p(r_1, \dots, r_k) \neq 0] \leq$$

$$\Pr_{r_{k+1}}[f_1(r_{k+1}) = 0],$$

where  $f_1(r_{k+1})$  is obtained after substituting the values  $r_1, \dots, r_k$  for  $x_1, \dots, x_k$ . Note that after such substitutions, we obtain a univariate polynomial of degree at most  $d_1$ . Hence,

$$\Pr_{r_{k+1}}[f_1(r_{k+1}) = 0] \leq d_1/|S|.$$

For the second summand, we have

$$\begin{aligned} & \Pr[f(r_1, \dots, r_{k+1}) = 0 \mid p(r_1, \dots, r_k) = 0] * \Pr[p(r_1, \dots, r_k) = 0] \leq \\ & \Pr[p(r_1, \dots, r_k) = 0] \leq \\ & (d - d_1)/|S|, \end{aligned}$$

where the last inequality is by the Inductive Hypothesis.

Putting everything together, we get

$$\Pr[f(r_1, \dots, r_{k+1}) = 0] \leq d_1/|S| + (d - d_1)/|S| = d/|S|,$$

as promised. □

### 3.2 Applications of the Schwartz-Zippel lemma

First, let's go back to our problem of testing whether two given arithmetic formulas  $f(x_1, \dots, x_n)$  and  $g(x_1, \dots, x_n)$  are equivalent. Let  $d_1$  and  $d_2$  be the total degrees of  $f$  and  $g$ , respectively. Consider the new polynomial  $h = f - g$ , of total degree at most  $d = \max\{d_1, d_2\}$ . Let  $S = \{1, 2, 3, \dots, N\}$ , for  $N = 2d$ . Then, testing whether  $f$  and  $g$  agree on a random  $n$ -tuple of elements from  $S$  is the same as testing whether  $h$  is zero on that tuple. If  $f \neq g$ , then  $h \neq 0$ , and by the Schwartz-Zippel lemma, the probability of picking a root for  $h$  is at most  $d/|S| = 1/2$ .

On the other hand, if  $f \equiv g$ , then  $h \equiv 0$ , and so  $h$  is zero on every point. Thus, we have a randomized polytime algorithm for testing if two arithmetic formulas are equivalent over integers. This algorithm has one-sided error, bounded from above by  $1/2$ .

In the above argument, we didn't say what the degrees  $d_1$  and  $d_2$  were. Let us define an arithmetic formula as a tree whose leaves are labeled by variables  $(x_1, \dots, x_n)$  or constants, and whose inner nodes are labeled by  $+$  or  $*$ . It is easy to prove by induction on the size of the formula, that a formula of size  $s$  can have degree at most  $s$ . Here the size of the formula is defined to be the number of leaves in the formula.

Here is the proof. For  $s = 1$ , the degree of the formula is either 0 or 1; so we're done. Let  $f = p_1 + p_2$ , where the size of  $f$  is  $s$ , and the sizes of  $p_1$  and  $p_2$  are  $s_1$  and  $s_2$ , respectively. Note that  $s = s_1 + s_2$ . Then by the Inductive Hypothesis, the degrees of  $p_1$  and  $p_2$  are at most  $s_1$  and  $s_2$ , respectively. Hence, the degree of  $f$  is at most  $\max\{s_1, s_2\} \leq s$ . The case of  $f = p_1 * p_2$  is similar - the degree of  $f$  is at most  $s_1 + s_2 = s$ .

So, when given an arithmetic formula in  $x_1, \dots, x_n$  of size  $m$ , we can always upperbound the degree of the polynomial computed by this formula by  $m$ . Then we can choose a set  $S = \{1, 2, \dots, 2m\}$ , from which we'll randomly pick values for the variables. The Schwartz-Zippel lemma guarantees that our error probability will be less than  $m/(2m) = 1/2$ .

One important point. Once we picked random values for the variables of our polynomial  $f(x_1, \dots, x_n)$  and started evaluating our polynomial, we may obtain big intermediate numbers. For example, if the degree of  $f$  is  $m$ , and if a largest element in  $S$  is  $2m$ , then the final value of  $f$  may be as big as  $(2m)^m$ . However, this value can be written down using  $O(m \log m)$  bits only, which is polynomial in the input size  $m$ .

What happens if we are given an arithmetic *circuit* rather than a formula? The difference now is that instead of a tree, we have a graph (DAG). Now, the degree of the polynomial given by an arithmetic circuit of size  $m$  can be much bigger than  $m$  - think of doing “repeated squaring” by having a sequence of multiplication gates  $g_1, \dots, g_m$  such that the inputs to gate  $g_{i+1}$  are two copies of the output from gate  $g_i$ ; then each such gate squares the degree of the polynomial, so that after  $m$  gates, the degree can be  $2^m$ , exponential in the size of the arithmetic circuit. However, this is the worst possible. Again, a simple argument by induction shows that the degree of a polynomial computed by an arithmetic circuit of size  $m$  is at most  $2^m$ .

This degree is big, and if we start evaluating a polynomial of degree  $2^m$  (even on small numbers), the final result may be as big as  $2^{2^m}$ , whose bit-size is at least  $2^m$ , exponential in the circuit size. We cannot hope to even write down such huge number in time polynomial in  $m$ . So what do we do?

The trick is to use modular arithmetic! We just pick a random prime  $p$  in the interval  $[1..2^{2^m}]$ , and perform all our calculations modulo that prime  $p$ . What is the probability that we are unlucky to pick a prime  $p$  that divides the value of circuit on some random inputs? First of all, let's pick the set  $S = \{1, 2, \dots, 2^{2^m}\}$ . The biggest possible value of our polynomial, given the inputs from  $S$ , will be  $v = (2^{2^m})^{2^m}$ . This number has at most  $2m2^m$  possible prime divisors. The interval  $[1..2^{2^m}]$  has at least  $2^{2^m}/(2m \ln 2) \geq 2^{1.5m}$  primes (by the Prime Number Theorem). Therefore, the probability of picking a prime divisor of the number  $v$  is at most  $2m2^m/(2^{1.5m}) \leq 2^{-0.4m} \ll 1$ , is very small. Thus, our randomized algorithm, modified to do modular arithmetic, is still very likely to be correct. But now this algorithm is guaranteed to run in polytime because all intermediate values  $\pmod p$  can be written down using only  $2m$  bits.