Lecture Notes 4

13 A Careful Proof of $\mathcal{BPP} \subseteq \mathcal{P}/\text{poly}$

We supply the missing piece to the proof of Theorem 4 in section 11 that $\mathcal{BPP} \subseteq \mathcal{P}/\text{poly}$. Namely, if $L \in \mathcal{BPP}$, there exists a ppTM M' recognizing L with error probability less than 2^{-n} for all inputs x of length n.

Lemma 1 *Let $L \in \mathcal{BPP}$. There is a probabilistic polynomial-time Turing machine M' such that*

1. $\forall x \in L, \Pr[M'(x) = 1] > 1 - 2^{-|x|}$;
2. $\forall x \notin L, \Pr[M'(x) = 0] > 1 - 2^{-|x|}$.

Proof: By definition of $\mathcal{BPP}$, there exists a ppTM M that recognizes L with error probability at most $1/3$ for each input x. We can assume without loss of generality that M always outputs either 0 or 1.

Let r be an odd positive integer to be determined later. The ppTM M' on input x of length n runs $M(x)$ a total of r times and gives as its output the majority value among the multiset of r outputs from M. We use the Chernoff bound (see lecture 1)

$$\Pr\left[\left|\frac{\sum X_i}{r} - p\right| \geq \varepsilon\right] \leq 2e^{-\frac{\varepsilon^2 r}{2p(1-p)}}. \tag{1}$$

plus additional probabilistic reasoning to bound the error probability of M'.

Define a random variable:

$$X_i = \begin{cases} 1 \text{ if the } i^{\text{th}} \text{ run of } M(x) \text{ gives the wrong answer} \\ 0 \text{ if the } i^{\text{th}} \text{ run of } M(x) \text{ gives the right answer} \end{cases}$$

The number of wrong answers in computing $M'(x)$ is $\sum X_i$, so $M'(x)$ gives the wrong answer iff $\sum X_i > r/2$.

We now calculate an upper bound on the error probability for $M'(x)$. Let p be the probability that $M(x)$ gives the wrong answer.[1] By assumption, $p \leq 1/3$.

$$\begin{aligned} \Pr[M'(x) \text{ gives wrong answer}] &= \Pr\left[\sum X_i > \frac{r}{2}\right] && (2) \\ &= \Pr\left[\left(\frac{\sum X_i}{r} - p\right) > \frac{1}{2} - p\right] && (3) \\ &\leq \Pr\left[\left(\frac{\sum X_i}{r} - p\right) > \frac{1}{6}\right]. && (4) \\ &\leq \Pr\left[\left|\frac{\sum X_i}{r} - p\right| > \frac{1}{6}\right]. && (5) \end{aligned}$$

[1] We do not assume that p is the same for all x.

Equation 4 follows since $1/2 - p \geq 1/6$, and equation 5 follows since the absolute value brackets only increase the cardinality of the event.

Taking $\varepsilon = 1/6$ and using the fact that $p(1-p) < 2/9$, equation 1 yields

$$\Pr\left[\left|\frac{\sum X_i}{r} - p\right| > \frac{1}{6}\right] \leq 2e^{-r/16}. \tag{6}$$

It is easily verified that

$$2e^{-r/16} < 2^{-n} \tag{7}$$

for all $r > 16(1+n)/\log_2 e$.

We now fix r to be the least odd positive integer satisfying equation 7. Combining equations 2–7 gives the desired error bound:

$$\Pr[M'(x) \text{ gives wrong answer}] < 2^{-n}. \tag{8}$$

Moreover, M' runs in polynomial time since M does and $r = O(n)$. This establishes the lemma. ■

14 One-Way Functions

One-way functions are at the heart of cryptography. Intuitively, a function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if it is "easy" to compute and "hard" to "invert". We next give precise technical definitions of these concepts.

14.1 Strongly one-way functions

Definition: The function f is *strongly one-way* if it satisfies the following conditions:

1. f is computable in polynomial time.
2. For all probabilistic polynomial-time algorithms A', all positive polynomials $p(\cdot)$, and all sufficiently large n,
$$\Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}. \tag{9}$$

Intuitively, condition 2 says that any algorithm A' that attempts to invert f has success probability less than one over any positive polynomial for all but finitely many n. Moreover, this fact remains true even when A' is given additional information about the inverse that it is trying to find.

The notation of equation 9 is very compressed and deserves some explanation. First of all, U_n is the uniformly distributed random variable on binary strings of length n. Each occurrence of U_n refers to the same length-n random string x, so each occurrence of $f(U_n)$ refers to same random string $y = f(x)$. While x is uniformly distributed among all strings of length n, y is not so distributed. We are not assuming at this point that f is length preserving or that all strings x of length n map onto y's of the same length. Moreover, f is not assumed to be one-to-one, so $\Pr[f(U_n) = y] = m2^{-n}$, where m is the number of length n strings that f maps to y.

Because we don't assume f is one-to-one, the notation $f^{-1}(y)$ refers to a set rather than to a particular string. By definition,

$$f^{-1}(y) = \{z \mid f(z) = y\}.$$

This set is called the *pre-image* of y. Note that if $y = f(x)$, then $x \in f^{-1}(y)$, but there may be other strings in the pre-iamge as well.

With this background, we see that the event within the square brackets of equation 9 is the event that A' succeeds in outputting some member of the pre-image $f^{-1}(y)$ for a randomly chosen string y as described above. Condition 2 then says that this probability that this event occurs is negligible (according to the definition of "negligible" in section 10). Note that A' is not required to output the string x that was used to generate y; any string in the pre-image will do.

Note also that A' is told the length of x as its second input. Even with this additional help, A' has only negligible success probability. This is done to prevent functions from being considered one-way only because their inverses are too long to output in a polynomial amount of time, e.g., the function $g(x) = \ell$, where ℓ is the length of x written as a binary number. For example, $g(111010111) = 1001$ since $|111010111| = 9$, and $(1001)_2 = 9$.

14.2 Two obvious algorithms

Here are two obvious algorithms for inverting a function f.

1. **Random guess.** $A_1(y, 1^n)$ outputs U_n, a uniformly distributed random string of length n. This is correct with probability at least 2^{-n} since there is a string x of length n such that $f(x) = y$, and A_1 has one chance in 2^n of finding it. It will be correct with probability strictly greater than 2^{-n} if there is more than one string of length n that f maps to y. In particularly, for the function f that maps every string to a constant c, A_1 will be correct with probability 1.

2. **Constant output.** $A_2(y, 1^n)$ outputs the string $x_0 = 0^n$ for every input of length n. This also is correct with probability at least 2^{-n} but for a different reason than A_1. Let $y_0 = f(x_0)$. Then $A_2(y)$ is correct when $y = y_0$. Since x is chosen uniformly at random from length-n strings, the probability that $y = y_0$ is at least 2^{-n}. As before, the probability will be greater if there is more than one string of length n that f maps to y_0. In particular, for the constant function f that maps every string to y_0, A_2 will be correct with probability 1.

Of course, to be strongly one-way, every p.p.t. algorithm A' to invert f must have only a negligible probability of success, not just these two algorithms. But these examples provide a lower bound on what we can expect from a one-way function. In particular, no function can have an inversion success probability smaller than 2^{-n}.

14.3 Weakly one-way functions

We now define a much weaker version of a one-way function. While uninteresting for cryptographic purposes, we will show that the existence of weakly one-way functions implies the existence of strongly one-way functions.

Definition: The function f is *weakly one-way* if it satisfies the following conditions:

1. f is computable in polynomial time.

2. There exists a positive polynomial $p(\cdot)$ such that for all probabilistic polynomial-time algorithms A' and all sufficiently large n,

$$\Pr[A'(f(U_n), 1^n) \notin f^{-1}(f(U_n))] > \frac{1}{p(n)}. \tag{10}$$

Note the subtle change in the order of quantifiers and in equation 10. This says that any algorithm A' that attempts to invert f will take a noticeable amount of the time on inputs of length n for all sufficiently large n.

A function $\mu(n)$ is *noticeable* if $\mu(n) > 1/p(n)$ for some fixed positive polynomial $p(\cdot)$ and all sufficiently large n. A function cannot be both noticeable and negligible, but it is possible for a function to be neither. For example, the function $\mu(n) = n \bmod 2$ that equals 1 on all odd numbers and 0 on all even numbers is neither negligible nor noticeable. It is not negligible because it fails to be smaller than $1/p(n)$ for infinitely many n, where $p(n) = 2$. It is not noticeable because it fails to be larger than $1/q(n) > 0$ for infinitely many n and all positive polynomials $q(n)$.