INFR11102: Computational Complexity 31/10/2019

Lecture 14: Randomised computation

Lecturer: Heng Guo

1 Randomised computation

The standard class for efficient randomised computation is called BPP (bounded-error probabilistic polynomial-time).

Definition 1. *A language L is in* BPP *if there exists a PTM P and a polynomial $p(\cdot)$, such that P runs in time $p(n)$ and*

- *if $x \in L$, then* $\Pr[P(x) = 1] \geq 3/4$*;*
- *if $x \notin L$, then* $\Pr[P(x) = 1] \leq 1/4$.

In other words, the requirement is that $\Pr[P(x) = L(x)] \geq 3/4$. The constant $3/4$ in Definition 1 is not essential, as long as it is strictly larger than $1/2$. This is captured by a notion called "amplification". On the other hand, if it is $1/2$, then the class defined would be (under standard assumptions) more powerful. That class is called PP, probabilistic polynomial-time. Historically PP is defined earlier than BPP, and as it turns out, PP is the *wrong* definition for efficient randomised computation.

We also have the "one-sided" error version of BPP, called RP (randomised polynomial-time).

Definition 2. *A language L is in* RP *if there exists a PTM P and a polynomial $p(\cdot)$, such that P runs in time $p(n)$ and*

- *if $x \in L$, then* $\Pr[P(x) = 1] \geq 3/4$*;*
- *if $x \notin L$, then* $\Pr[P(x) = 1] = 0$.

In other words, an RP algorithm never errs if $x \notin L$, but it may reject some $x \in L$.

The complementary class coRP is defined in the usual way:

$$\texttt{coRP} := \{L \mid \overline{L} \in \texttt{RP}\}.$$

There is also a "zero-error" version, called ZPP.

Definition 3. *A language L is in* ZPP *if there exists a PTM P and a polynomial $p(\cdot)$, such that the expected running time of P is at most $p(n)$ and if P halts on input x, $P(x) = L(x)$.*

In fact $\mathtt{ZPP} = \mathtt{RP} \cap \mathtt{coRP}$. To show $\mathtt{ZPP} \subseteq \mathtt{RP}$, we just run the $\mathtt{ZPP}$ algorithm, and answer 0 if the we have used, say, 4 times the expected running time. The probability for that to happen is at most 1/4, due to Markov's inequality.

Lemma 1 (Markov's inequality)**.** *Let X be a non-negative random variable. For any $a > 0$,*

$$\Pr[X \geq a] \leq \frac{\mathbb{E}\, X}{a}.$$

Proof. Let I be an indicator variable for the event $X \geq a$. Namely

$$I = \begin{cases} 1 & \text{if } X \geq a; \\ 0 & \text{if } X < a. \end{cases}$$

Then, $\mathbb{E}\, I = \Pr[X \geq a]$. Note that $I \leq \frac{X}{a}$ in all circumstances. (This is called stochastic domination.) Then

$$\Pr[X \geq a] = \mathbb{E}\, I \leq \mathbb{E}\left[\frac{X}{a}\right] = \frac{\mathbb{E}\, X}{a}. \qquad \square$$

By a similar argument, $\mathtt{ZPP} \subseteq \mathtt{coRP}$, implying that $\mathtt{ZPP} \subseteq \mathtt{RP} \cap \mathtt{coRP}$.

On the other hand, the $\mathtt{ZPP}$ algorithm for a language in $L \in \mathtt{RP} \cap \mathtt{coRP}$ is to run both the $\mathtt{RP}$ and the $\mathtt{coRP}$ algorithms simultaneously. If one correct answer is obtained, then we accept it. Otherwise we restart and do it again. It is easy to check that the restarting probability is at most 1/4, and thus the total expected running time T satisfies

$$T \leq \frac{3}{4}T_0 + \frac{1}{4}(T + T_0),$$

where T_0 is the maximum running time of the $\mathtt{RP}$ and $\mathtt{coRP}$ algorithms. Solving it gives us $T \leq 4T_0/3$, which is still a polynomial.

2 Amplification of BPP

Amplification for $\mathtt{RP}$ algorithm is straightforward — just run it many times.

For $\mathtt{BPP}$, the constant 3/4 in Definition 1 can be replaced with any constant $> 1/2$. In fact, even just barely larger than 1/2 is enough (but not 1/2). This is captured by the following amplification theorem. In fact, the error probability can be amplified to exponentially small.

Theorem 2. *Let L be a language and suppose that there is a polynomial-time PTM P and a constant c such that for every x, $\Pr[P(x) = L(x)] \geq 1/2 + |x|^{-c}$. Then for every constant $d > 0$, there is a polynomial-time PTM P' such that for every x, $\Pr[P'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.*[1]

[1] $\exp(x)$ is a shorthand for e^x.

The idea is simple: P' will just run P many times and then take a majority vote. In order to bound the error probability, we will need the so-called "Chernoff bound".

Lemma 3 (Chernoff Bounds)**.** *Let $X_1, \cdots, X_n$ be mutually independent, identically distributed (i.i.d.) random variables over $\{0,1\}$, with probability p to be 1. For any $0 < \varepsilon \le 1$,*

$$\Pr\left[\sum_{i=1}^{n} X_i \ge (1+\varepsilon)pn\right] \le \exp\left(-\frac{\varepsilon^2 pn}{3}\right);$$
$$\Pr\left[\sum_{i=1}^{n} X_i \le (1-\varepsilon)pn\right] \le \exp\left(-\frac{\varepsilon^2 pn}{3}\right).$$

Note that the form above is not the strongest possible, but it is useful, neat, and easy to remember. A more general form and proofs can be found in [AB09, Theorem A.14]. Lemma 3 is proved via applying Markov's inequality, Lemma 1, on a suitable exponential random variable.

What Lemma 3 is saying is that if we have a number of independent random variables, then their sum cannot deviate from the mean by too much. In particular, it implies that with constant probability, the deviation is within $O(\sqrt{n})$.

Proof of Theorem 2. As said earlier, P' will run P for $s = 24n^{2c+d}$ times and then take a majority vote, where $n = |x|$ is the input length. Let X_i be the output of the ith run and $X = \sum_{i=1}^{s} X_i$. Let $p = 1/2 + n^{-c}$. If the majority vote went wrong, then roughly $n^{-c}s$ of all these runs are wrong, which would have very small probability due to Lemma 3 since we set $s \gg n^{2c}$.

To be more precise, let us assume that $x \in L$. (The other case is completely symmetric.) Then, the output is wrong, if and only if $X < s/2$. Namely, we can set $\varepsilon = n^{-c}/2$ in Lemma 3, so that $(1-\varepsilon)p > 1/2$, and

$$\begin{aligned}\Pr[X < s/2] &\le \Pr[X < (1-\varepsilon)ps] \\ &\le \exp\left(-\frac{n^{-2c}(1/2+n^{-c})s}{12}\right) \\ &\le \exp\left(-\frac{n^{-2c}s}{24}\right) = \exp\left(-n^d\right).\end{aligned}$$

This finishes the proof. □

3 Examples

Can we solve Sat using randomised algorithms? A natural attempt is then to randomly assign values to variables, and test if the assignment is satisfying. This will not work because the probability that a random assignment is satisfying is exponentially small. Thus, although we will never accept an unsatisfying formula, we will not accept a satisfying formula either until exponential time is spent.

The key to design a randomised algorithm is thus to find a good trial that will succeed with good probability.

3.1 Prime number testing

Recall the problem PRIMES, which asks to determine whether an input integer (in binary) is prime. Note that PRIMES can be solved in polynomial-time [AKS04], but the deterministic algorithm is very complicated. Here we will see a couple of efficient randomised algorithms that are used in practise.

We may assume the number to test is odd throughout. By Fermat's little theorem, we know that

$$a^{p-1} \equiv 1 \mod p \tag{1}$$

if p is a prime number and $1 \le a < p$. Thus, for an input n, we may randomly pick a from $1, 2, \cdots, n-1$. If $a^{n-1} \not\equiv 1 \mod n$, then we know that n is not a prime. However, this test is not efficient in that the success probability is small.

A more efficient test is to test $a^{\frac{n-1}{2}}$. By (1), it must be that $a^{\frac{p-1}{2}} \equiv \pm 1 \mod p$ for $a \in \{1, 2, \cdots, p-1\}$ and a prime p. It is an easy fact that

$$a^{\frac{p-1}{2}} \equiv 1 \mod p \qquad \text{if and only if} \qquad \exists q \text{ s.t. } q^2 \equiv a \mod p. \tag{2}$$

The latter fact can be efficiently decided using the law of quadratic reciprocity, and thus we may randomly pick a and verify (2). Solovay and Strassen [SS77] showed that a single run of this test succeeds with $> 1/2$ probability. Thus, if n passed k such tests, then we are sure that n is a prime with probability $1 - 2^{-k}$.

An even more efficient test is the Miller-Rabin test. We go back to (1). We know that $p-1$ must be an even number, and assume that $p - 1 = 2^k t$. Then (1) can be rewritten as

$$(a^t - 1)(a^t + 1)(a^{2t} + 1) \cdots (a^{2^{k-1}t} + 1) \equiv 0 \mod p. \tag{3}$$

Thus it must be that either $a^t \equiv 1 \mod p$, or $\exists i \le k-1$, $a^{2^i t} \equiv -1 \mod p$. We say that a is a *Miller-Rabin witness* for an odd number n if all factors in (3) are non-zero. Namely, for $n - 1 = 2^k t$, $a^t \not\equiv 1 \mod n$ and $\forall i \le k-1$, $a^{2^i t} \not\equiv -1 \mod n$.

Miller [Mil76] proposed to use the idea above to do primality testing, and but his original algorithm requires generalised Riemann hypothesis. Rabin [Rab80] and Monier [Mon80] independently showed that if n is not a prime, then there are more than $\frac{3}{4}$ fraction Miller-Rabin witnesses amongst natural numbers less than n. Thus, a single run of the algorithm succeeds with probability at least $3/4$, if we pick a uniformly at random. If n passed k such tests, then we are sure that n is a prime with probability $1 - 4^{-k}$.

All of the algorithms above are coRP algorithms for PRIMES as they will always accept primes but sometimes not reject composites.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976.

[Mon80] Louis Monier. Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theor. Comput. Sci.*, 12:97–108, 1980.

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128 – 138, 1980.

[SS77] Robert Solovay and Volker Strassen. A fast Monte-Carlo test for primality. *SIAM J. Comput.*, 6(1):84–85, 1977.