6.841 Advanced Complexity Theory March 7, 2005

Lecture 10

Lecturer: Madhu Sudan *Scribe: Rafael Pass*

1 Randomness and Computation

Recall the strong form of the Turing-Church hypothesis: *Every physically realizable form of computation is simulated by a Turing Machine with a polynomial slowdown.*[1] Clearly this hypothesis is not a mathematical statement, and indeed Church and Turing "argued" for its validity by simply by comparing the Turing model of computation with other models. Soon, this hypothesis meet criticism from the physics community, raising the question of wheter randomized computation, such as for example the Brownian motion, could be simulated by a Turing machine. This was followed by the discovery of problems that themselves didn't refer to randomness, but the only known algorithms to solve them relied on randomization.

Consider, for example the following problems. Each of them has a randomized algorithm, although no deterministic algorithms are known.

1. *Given a n-bit integer N, find a prime $p \in [N+1, N^2]$.* A simple randomized algorithm for this problem is to pick a random number in the range and test if it is a prime. However, no deterministic algorithms are known.
2. *Given a n-bit prime p and an integer a, find α such that $\alpha^2 = a \pmod p$.*
3. *Given k matrices $M_1, ..., M_k$, where each $M_i \in Z^{n\times n}$, find integers (or even decide whether there exists) $\gamma_1, ..\gamma_k$ such that*
$$\sum_{i=1}^{k} \gamma_i M_i$$
is non-singular (i.e., such that the determinant is non-zero).
4. *Given algebraic circuits (i.e., with addition and multiplication gates with two inputs) C_1, C_2 over $\mathcal{Z}$, decide whether they are computing the same function.* Note that in the case of *boolean* circuits this problem is **NP**-complete. Over the integers, however, randomized algorithms exists.

Note that none of the above-mentioned problems mention randomness in their description, yet all known (efficient) algorithms to solve them uses it.

1.1 Some Facts on Why Algebra/Number Theory Problems are Amenable to Randomization

All the above-mentioned problems are algebraic or number-theoretic in nature. We describe a couple of facts explaining why such problem seem amenable to randomized algorithms.

1. *Let G be a finite group, and let H be a subgroup of G. Then $\frac{|H|}{|G|} \leq \frac{1}{2}$, or $H = G$.* This means that elements of $G - H$ can be found "easily" if sampling from G is "easy"). This fact explains why randomization helps in finding a solution to Problem 2, above.

[1]Note that it is the polynomial slowdown requirement that makes it the "strong" hypothesis.

2. *Prime Number Theorem: the number of primes in the intervall* $[1..n]$ *is "roughly"* $\frac{N}{lnN}(1+2^{-o(\sqrt{\log n})})$. This fact explains why randomization helps in Problem 1.

3. *Let* $N \leq 2^n$*, and let* $p_1, .., p_{2n}$ *be relatively prime integers. Then* $N \neq 0 \pmod{p_i}$ *with probability* $1/2$*, if* $i \in_R \{1, .., 2n\}$. This follows from the unique factorization of N and that the fact that N is the product of at most n primes.

4. *[DeMillo-Lipton, Schwatrz, Zippel] Lemma: Let* $p \neq 0$ *be a polynomial of degree* d *in* n *variables. Then*

$$\Pr_{\alpha_1,...,\alpha_n \in_R S}[p(\alpha_1, ..., \alpha_n) = 0] \leq \frac{d}{|S|}$$

   Note that this property was used in the circuit lower bound by Razborov-Smolensky (see Lecture 7). Also note that this lemma can be used to solve Problem 3 (and also Problem 4, although some additional properties are also needed).

2 A Randomized Algorithm for Finding Square roots mod p

We show how to give a randomized algortihm for Problem 2, i.e., finding square roots modulo a prime p. Towards this goal, we show a specialization of Berlekamp's factoring algorithm for $Z_p[x]$ (i.e., polynomials over Z_p) [1, 2]. In fact, in order to find the sqaure root of an element a, mod p, we consider the following polynomial:

$$x^2 - a = 0$$

Note that a factorization $x^2 - a = (x - \alpha)(x + \alpha)$ yields a square root α to a. Below we show a randomized algorithm to factor polynomials of this kind.

First, consider the following polynomial:

$$\prod_{\beta \in Z_p} (x - \beta) = x^p - x$$

(because every $\beta \in Z_p$ is a root to $x^p - p$, since $\beta^p = \beta$ by Fermat's theorem). Therefore,

$$x^2 - a | x^p - x$$

Since $p - 1$ is even,

$$x^p - x = x(x^{p-1} - 1) = x(x^{(p-1)/2} - 1)(x^{(p-1)/2} + 1)$$

Suppose that we are "lucky" and that $x - \alpha | x^{(p-1)/2} - 1$ and $x + \alpha | x^{(p-1)/2} + 1$. In this event, the factorization of $x^2 - a$ can be found by computing $gcd(x^2 - a, x^{(p-1)/2} - 1)$, and we are done.

When does this lucky event happen? Recall that $x - \alpha | x^{(p-1)/2} - 1$ if $\alpha^{(p-1)/2} = 1 \; mod p$. Likewise $x + \alpha | x^{(p-1)/2} + 1$ if $\alpha^{(p-1)/2} = 1 \; mod\, p$. By combing these equation we thus get that the "lucky" event happens when $(-1)^{(p-1)/2} = -1 \; mod\, p$, i.e., when $p = 3 \; mod\, 4$.

Adding randomization. In order to make the above technique work on every p we introduce the following "randomization"-twist. Rather than trying to factor the function $x^2 - a$, we will instead consider the function $(x+\gamma)^2 - a$, where γ is a random integer in Z_p. Clearly the factorization $(x+\gamma)^2 - a = (x-\delta)(x-\epsilon) = ((x+\gamma)+\alpha)((x+\gamma)-\alpha)$ yields the factorization to $x^2 - a = (x+\alpha)(x-\alpha)$. We proceed as before and note that in the "lucky" event that $x+\gamma+\alpha | x^{(p-1)/2} - 1$ and $x+\gamma-\alpha | x^{(p-1)/2}+1$ (or vice versa) we can recover α by computing the greatest common divisor of $(x+\gamma)^2 - a$ and $x^{(p-1)/2} - 1$. We call such γ "good".

Analyzing the probability of getting a "good" γ. The "lucky" events happens when $(\alpha+\gamma)^{(p-1)/2} = 1$, and $(\alpha-\gamma)^{(p-1)/2} = -1$, or when $(\alpha+\gamma)^{(p-1)/2} = -1$, and $(\alpha-\gamma)^{(p-1)/2} = 1$. Combining these equations we conclude that a γ is "good" when

$$\left(\frac{\gamma-\alpha}{\alpha+\gamma}\right)^{(p-1)/2} = -1$$

which is equivalent to

$$\left(1 - \frac{2\alpha}{\alpha+\gamma}\right)^{(p-1)/2} = -1$$

Note that for a random $\gamma \in Z_p$, $\alpha+\gamma$ is a random element in Z_p, which means that $\frac{1}{\alpha+\gamma}$ is a random element in Z_p^*. Therefore, $1 - 2\frac{\alpha}{\alpha+\gamma}$ is a "random"[2] element in Z_p. This in turn means that with probability roughly 1/2, a random γ will be "good" and we will be able to find the factorization.

Summing up. We have thus shown that in order to factor the polynomial $x^2 - a$, the following algorithm will succeed with probability roughly 1/2:

1. Pick a random element $\gamma \in Z_p$.
2. Compute $gcd((x^2+\gamma)^2 - a, x^{(p-1)/2} - 1)$.
3. If the factor obtained is non-trivial and of the form $x - \delta$, output $\delta - \gamma$.

We finally note that this algorithm also generalizes to arbitrary polynomials.

3 Models of Randomized Computation

We now address the question of defining randomized computation. One approach would be to allow "random"-states in a Turing Machine. Another approch, which is the one we follow, is to consider 2-input Turing Machines $M(\cdot,\cdot)$ which run in polynomial time. Randomized (probabilistic) computation is then defined by considering the output of $M(x,y)$ on input an instance x, and a random string y.

[2]Actually, it is only "almost" random since it will never be 1.

Deciding Languages by Probabilistic Machines. In order to define what it means for a probabilistic machine to decide a language L, consider the following notions:

- We say that $M(\cdot, \cdot)$ has *Completeness* c if for all instances $x \in L$,

$$\Pr_{y \in_R \{0,1\}^*}[M(x, y) = accept] \geq c$$

- We say that $M(\cdot, \cdot)$ has *Soundness Error* s if for all instances $x \notin L$,

$$\Pr_{y \in_R \{0,1\}^*}[M(x, y) = accept] \leq s$$

If M satisfies both the above conditions we say that M decides L with completeness c and soundness s.

We can now define the class BPP (Bounded Probabilistic Polynomial Time) as the class of languages that are decidable by machines that have a "gap" between the completeness and soundness. More precisely,

Definition 1 *We say that a language $L \in BPP$ if there exists a polynomial time machine $M(\cdot, \cdot)$ and constants $\epsilon > 0$,c, s,$c > s + \epsilon$, such that M decides L with completeness c and soundness s.*

We note that an alternative (but equivalent) definition is to require that the gap is of some "large" constant size. This follows from the fact that the "gap" can be *amplified* by repeating the decision procedure multiple times and outputting the *majority* of the decisions (and relying on the Chernoff bound). More details will follow in the next lecture.

Definition 2 *We say that a language $L \in BPP$ if there exists a polynomial time machine $M(\cdot, \cdot)$ such that M decides L with completeness* $2/3$ *and soundness* $1/3$.

In fact, using the same amplification technique it can be seen that the BPP can be characterized by languages decidable with completeness $1 - 2^{-n}$ and soundness 2^{-n}.

Zero Probability Error. Note that the Factoring Algorihm described in the Section 2 has the property that it either outputs `fail` or always gives the right answer. Namely, conditionned on not outputting fail, the completeness is 1 and soundness 0. Alternatively, the algorithm has an *expected* polynomial running time but always outputs the right answer. The class of languages having this property is denoted ZPP (Zero Probability Polynomial time).

One-sided Error. We might also consider languages that are decided with a one-sided error, i.e.,

- Completeness $> \epsilon$. Soundness 0. This class is called RP (randomized polynomial time).
- Completeness $= 1$. Soundness $< 1 - \epsilon$. This class is called $coRP$.

Relationship between RP and NP. We note that $RP \subseteq NP$ and $coRP \subseteq coNP$ We show the former statement. In fact, recall that $L \in NP$ if there exists a polynomial time machine M such that

- for all $x \in L$, there exists a string y such that $M(x, y)$ accepts. This is equivalent to saying that
$$\Pr_y[M(x, y) = \mathit{accept}] > 0$$
- for all $x \notin L$, for all strings y $M(x, y)$ rejects. This is equivalent to saying that
$$\Pr_y[M(x, y) = \mathit{accept}] = 0$$

Thus everything in RP is also in NP.

Relationship between the different classes of Probabilisitic Computation. By the definition if follows that P is contained in ZPP, which in turn is contained in RP and $coRP$. Finally, both RP and $coRP$ are contained in BPP. The relationship between RP and $coRP$ is unknown. What is known, though, is that if $BPP = P$ then we would have strong circuit lower bounds.

Analogous containments also hold for probabilistic logarithmic time computation. Namely, L is contained in ZL, which is contained in RL and $coRL$, which both are conatined in BPL. Also here the relationship between RL and $coRL$ is unknown. Nevertheless, over the last couple of months there has been much progress in this area of research, which seems to indicate that the question of whether $RL = L$ is within reach.

We end this section by noting the following fact, which is left as an exercise.

Fact 3 $ZPP = RP \cap coRP$

4 Promise Problems

Recall that computational problems for a languages L have the property that for all instances $x \in L$ the machine that decides the language outputs "YES", while for all instance $x \notin L$ it outputs NO. It is sometime convenient to relax this definition to *Promise Problems* (for more info, see the survey [3]). A promise problem is a pair of languages $L_{yes}, L_{no} \subseteq \{0,1\}^*$ such that $L_{yes} \cap L_{no} = 0$, that can be decided by a machine M in the following way:

- When $x \in L_{yes}$ then $M(x)$ outputs "YES".
- When $x \in L_{no}$ then $M(x)$ outputs "NO".
- Otherwise M can output anything.

In analogy with the above deterministic definition we can provide a definition of $PromiseBPP$ (which is a generalization, i.e., a superset, of BPP).

Definition 4 *We say that a promise problem* $\Pi = (L_{yes}, L_{no}) \in PromiseBPP$ *if there exists a polynomial time machine* $M(\cdot, \cdot)$ *such that*

- *When* $x \in L_{yes}$,

$$\Pr_{y \in_R \{0,1\}^*}[M(x,y) = accept] \geq 2/3$$

- *When* $x \in L_{no}$,

$$\Pr_{y \in_R \{0,1\}^*}[M(x,y) = accept] \leq 1/3$$

A Promise version of $(NP \cap coNP)$ In the same vein, we can define a promise version of $NP \cap coNP$, promise($NP \cap coNP$). Interestingly, the following fact can be shown concerning this class,

Fact 5 *If promise($NP \cap coNP$) $= P$, then $NP = P$.*

(We note that it is unknown if the statement "If $NP \cap coNP = P$, then $NP = P$" is true.) The above-mentioned fact is proved by considering the following complete problem for promise($NP \cap coNP$):

- $L_{yes} = \{\Phi, \Psi | \Phi \in SAT, \Psi \notin SAT\}$
- $L_{no} = \{\Phi, \Psi | \Phi \notin SAT, \Psi \in SAT\}$

It can be shown that if the above promise problem can be decided in polynomial time, then SAT can so as well. More precisely, assume that there exists a polynomial time machine M that decides the above promise problem. We show how to use M to decide if an instance $\Phi \in SAT$, by proceeding as follow. Invoke $M(\Phi_{x_0=0}, \Phi_{x_0=1})$. If M outputs 1, then let $a_0 = 0$ and otherwise $a_0 = 1$. Now fix $x_0 = a_0$ in Φ, and proceed analogously for $x_1, x_2, .., x_n$ to obtain $a_1, .., a_n$. Finally, output $\Phi(a_0, .., a_n)$. Note if $\Phi \notin SAT$ then $\Phi(a_0, .., a_n) = 0$. On the other hand, if $\Phi \in SAT$ then $a_0, .., a_n$ will be a satisfying assignment. This follows from the fact that in each iteration i, if $\Phi_{x_0=a_0,..,x_{i-1}=a_{i-1}} \in SAT$, then either $\Phi_{x_0=a_0,..,x_{i-1}=a_{i-1},x_i=0} \in SAT$, or either $\Phi_{x_0=a_0,..,x_{i-1}=a_{i-1},x_i=1} \in SAT$, or both. In the case that only one is satisfiable then M will give us the right answer. If both are satisfiable then *any* answer is fine.

References

[1] E.R. Berlekamp. "Factoring Polynomials Over Large Finite Fields", Math. of Computation, 24:713-735, 1970.

[2] D. Knuth. "The Art of Computer Programming. Volume 2", 1997.

[3] O. Goldreich. "On Promise Problems", 2005.