6.841 Advanced Complexity Theory March 12, 2007

Lecture 10: Randomized Computation

Lecturer: Madhu Sudan *Scribe: Punyashloka Biswal*

We will motivate and define the complexity classes used to describe randomized computation, and discuss a few of their basic properties.

1 Motivation

There are several natural problems that we know how to solve probabilistically (i.e., using randomness in some way), but not deterministically. Here are a few examples:

Prime in interval Given an n-bit integer N, we wish to find a prime $p \in [N+1, 2N]$. A variant of this problem asks for a prime in the interval $[N+1, 2^{(\log N)^\epsilon}]$.

To solve this probabilistically, pick p uniformly from $[N+1, 2N]$. Test if it is prime (this can be done deterministically), and if so, output it. If not, repeat. By the Prime Number Theorem, such a p is prime with probability $\geq \Omega(1/n)$, so that we have a polynomial number of trials in expectation.

Square roots in $\mathbb{Z}_p$ Given an n-bit prime p and an integer $a \in \mathbb{Z}_p$, we wish to find $\alpha \in \mathbb{Z}_p$ such that $\alpha^2 = a$. This is a special case of the problem of factoring polynomials over finite fields.

This problem is trivial for $p = 2$. When $p \cong 3 \pmod 4$, it can be solved deterministically: $\alpha = \pm a^{(p+1)/4}$. The key step in the case when $p \cong 1 \pmod 4$ is finding a quadratic non-residue q. The quadratic residues form a cyclic subgroup of index 2 in $\mathbb{Z}_p$, so there are $(p-1)/2$ such q's. This means that they are very easy to find by randomized trials. Unfortunately, we know of no efficient way to find them deterministically.

Nonsingular matrix sum Given k matrices $M_1, \ldots, M_k \in \mathbb{Z}^{n \times n}$, we wish to find $\lambda_1, \ldots, \lambda_k \in \mathbb{Z}$ such that

$$\det\Big(\sum_{i=1}^{k} \lambda_i M_i\Big) \neq 0.$$

To solve this deterministically, pick $\lambda_i \in_U S$, where $S = [2n]$, and test if the determinant is zero in this case. If not, output these λ_is, and if not, then repeat.

To understand why this is an efficient algorithm, consider the degree-n polynomial $p(x_1, \ldots, x_n) := \det(\sum_{i=1}^{k} M_i x_i)$. By the Schwartz lemma, $\Pr[p(\lambda_1, \ldots, \lambda_k) = 0] \leq n/|S| = 1/2$.

Algebraic Circuit Identity Testing We are given an arithmetic circuit C over the integers with gates for addition, subtraction, and multiplication and inputs which are either variables or constants. We wish to test whether C computes the zero function.

Exercise: Give a randomized co-RP algorithm for ACIT. [1]

2 Computational assumptions

The notion of probabilistic computation seems 'realistic' in the sense that nature exhibits unpredictable behavior, and we should be able to extract some of this unpredictability in the form of random inputs to algorithms. However, the previous section shows that we can pose natural problems that we do not know how to solve without the aid of randomness. This conflicts with the strong form of the Turing-Church Hypothesis:

Strong Turing-Church Hypothesis Every physically realizable form of computation is simulatable on a Turing machine with only a polynomial time slowdown.

The two key features of this hypothesis are

1. Locality: A Turing machine has a finite-state controller. As a result, even though it has an infinite tape, its actions at any one moment are controlled by a finite, localized set of inputs.

2. Determinism: A Turing machine's actions are fully determined by its past.

Randomized computation preserves the former property, but not the latter. Guided by our sense that the physical world contains randomness, that modern-day computers can simulate randomized algorithms, and that we are unable to do without randomization for several natural problems[2] we are led to redefine the Turing-Church hypothesis as

Strong Turing-Church Hypothesis, probabilistic version Every physically realizable form of computation is simulatable on a *probabilistic* Turing machine with only a polynomial time slowdown.

It is instructive to contrast this with the situation in quantum computing. Quantum computation is another resource that nature provides us, which lets us perform computatations that are not known to be in P (such as factoring integers). However, present-day computers are unable to simulate quantum computation, so we are wary of making a corresponding change to the Turing-Church hypothesis.

[1] This is not entirely trivial, as demonstrated by the following example: let S be the circuit that maps $x \mapsto x^2$, and define $C = S \circ \cdots \circ S = S^{\circ n}$. Observe that C is a small circuit with small constant inputs, but it outputs a polynomial of degree exponential in n. This makes it impossible to determine whether it computes zero simply by sampling from a small set of values of x.

[2] If BPP = P, as some people believe, this addition is unnecessary and the original hypothesis stands.

3 Complexity classes

We could define complexity classes for randomized computation by extending classical Turing machines with randomized primitives (such as "toss a fair coin"). It turns out that it is better to adapt the two-tape definition of NP, which we repeat below for reference:

Definition 1 (Non-deterministic polynomial time, NP) *A language* $L \in \text{NP}$ *if there exists a 2-input Turing machine* $M(\cdot,\cdot)$ *with worst-case running time polynomial in the length of the first input* x*, such that*

(Completeness) $x \in L \Rightarrow \exists y, M(x,y)$ *accepts, and*

(Soundness) $x \notin L \Rightarrow \forall y, M(x,y)$ *rejects.*

Similarly, we have

Definition 2 (Randomized polynomial time, RP) *A language* $L \in \text{RP}$ *if there exists a 2-input Turing machine* $M(\cdot,\cdot)$ *and an efficiently sampleable distribution on* y *such that* M *has expected running time polynomial in the length of the first input* x *and*

(Completeness) $x \in L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] \geq 2/3$*, and*

(Soundness) $x \notin L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] = 0$*.*

Definition 3 (Complements of RP, co-RP) *A language* $L \in \text{co-RP}$ *if there exists a 2-input Turing machine* $M(\cdot,\cdot)$ *and an efficiently sampleable distribution on* y *such that* M *has expected running time polynomial in the length of the first input* x *and*

(Completeness) $x \in L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] = 1$*, and*

(Soundness) $x \notin L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] \leq 1/3$*.*

Definition 4 (Bounded-error probabilistic polynomial time, BPP) *A language* $L \in \text{BPP}$ *if there exists a 2-input Turing machine* $M(\cdot,\cdot)$ *and an efficiently sampleable distribution on* y *such that* M *has expected running time polynomial in the length of the first input* x *and*

(Completeness) $x \in L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] \geq 2/3$*, and*

(Soundness) $x \notin L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] \leq 1/3$*.*

Definition 5 (Zero-error probabilistic polynomial time, ZPP) *A language* $L \in \text{ZPP}$ *if there exists a 2-input Turing machine* $M(\cdot,\cdot)$ *and an efficiently sampleable distribution on* y *such that* M *has* ***expected*** *running time polynomial in the length of the first input* x *and*

(Completeness) $x \in L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] = 1$*, and*

(Soundness) $x \notin L \Rightarrow \Pr_y[M(x,y)\ \textit{accepts}] = 0$*.*

A note on expected running time In the first three of these definitions, it is possible to replace "expected running time" with "worst-case running time" if we are willing to relax the error parameters slightly; this also lets us compare BPP/RP/co-RP algorithms with deterministic ones on a more equal footing. As we shall see in the next section, all these definitions are robust across a wide range of values of the error parameters, so we do not lose any generality.

Definition 6 (RP, alternate) *A language $L \in$ RP if there exists a 2-input Turing machine $M(\cdot,\cdot)$ and an efficiently sampleable distribution on y such that M has worst-case running time polynomial in the length of the first input x and*

(Completeness) $x \in L \Rightarrow \Pr_y[M(x,y) \text{ accepts}] \geq 5/8$, *and*

(Soundness) $x \notin L \Rightarrow \Pr_y[M(x,y) \text{ accepts}] = 0$.

Proof To see why this definition implies the old one, suppose $L \in$ BPP and let M be a BPP Turing machine that takes a random tape in addition to x and decides L in expected time $p(|x|)$. We shall construct another Turing machine M' for L as follows: on input x, we run M on x and a random tape for $10p(|x|)$ time steps. If M does not terminate by this time, then reject.

By the Markov inequality, M fails to terminate in $10p(|x|)$ steps with probability $\leq 1/10$. Therefore, the probability that an $x \in L$ is not accepted is at most $2/3 - 1/10 \geq 5/8$.

The converse is trivially true after performing an amplification. ■

Unlike for BPP, RP, and co-RP, expected running time is crucial to the definition of ZPP and cannot be eliminated. For example, consider the the language

$$L = \{(p, a, b, c) \mid p \text{ is a prime}, a, b, c \in \mathbb{Z}_p, \exists \alpha \in [b, c] : \alpha^2 = a\}.$$

This is a decision version of the square root problem, and is easily shown to be of equivalent hardness. It has a ZPP algorithm but no known deterministic one.

However, we have a different characterization of ZPP in terms of other classes defined using worst-case time:

Proposition 7 ZPP = RP ∩ co-RP

Proof Let M and M' be RP and co-RP algorithms, respectively, for a language L that lies in the intersection. Run them in parallel. If M accepts first, then accept. If M' rejects first, then reject. One of these two must eventually happen, and in expected polynomial time.

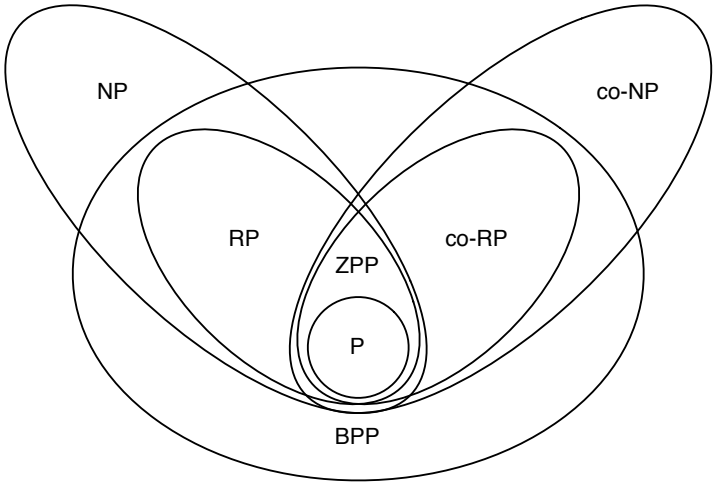
The converse is trivial. ■

Inclusions among randomized classes These definitions make it clear that RP ⊆ NP and co-RP ⊆ co-NP. We shall see in a later lecture that RP and co-RP lie within Σ_2^{P}. This gives us the picture below (Figure 1). Some complexity theorists believe that BPP = P ≠ NP ≠ co-NP.

4 Amplification

The error parameters used in the definitions from the previous section can actually be replaced with a wide range of different values without altering the complexity classes involved.

Figure 1: Known inclusions



For example, we can change "$\geq 2/3$" in the definition of RP to "$\geq 1/p(n)$" or "$\geq 1 - 2^{-p(n)}$" for any polynomial function p of the input length: take an RP machine M with error probability $1 - 1/p(n)$, and run it t times on independently chosen random y's. Accept if any of these runs accept. The error probability of this new Turing machine is $(1 - 1/p)^t$, which can be made $\leq e^{-q(n)}$ by setting $t = \Omega(p(n)q(n))$.

For BPP, we can replace the correctness statement "$\geq 1/2 + 1/p(n)$" with "$\geq 1 - 2^{-q(n)}$" and the soundness statement "$\leq 1/2$" with "$\leq 2^{-q(n)}$." The amplification argument is conceptually similar, but uses Chernoff bounds.