15-859(M): Randomized Algorithms **Lecturer:** Shuchi Chawla
Topic: Complexity classes, Identity checking **Date:** September 15, 2004
Scribe: Andrew Gilpin

2.1 Complexity Classes

In this lecture we will look at some randomized complexity classes: RP, co$-RP$, ZPP, and BPP. We begin with a (very brief) review of P and NP.

Pointers are given to the appropriate sections of Motwani & Raghavan, denoted M&R, where appropriate: for complexity classes, one can consult Section 1.5.2 of the book. A nice complexity resource is on the web at `http://www.complexityzoo.com/`. This site contains, at last count, definitions of 406 complexity classes, including brief discussions of relations between classes.

It is useful to recall the following basic definition:

Definition 2.1.1 *A language L is a set of finite strings over some fixed alphabet Σ; i.e., $L \subseteq \Sigma^*$. An algorithm is said to recognize L if on input $x \in L$, the algorithm outputs* Yes*, and on input $x \notin L$, it outputs* No.

The following classes are well-known:

P: **Polynomial time** A language L lies in P if there is a polynomial-time algorithm that recognizes L.

NP: **Non-deterministic Polynomial time** The class NP consists of all languages L which have witnesses that can be recognised by polynomial time. More formally, $L \in NP$ implies that there is some polynomial time-computable predicate $\mathcal{P}$, and a polynomial p such that

$$x \in L \qquad \iff \qquad \exists y \; \mathcal{P}(x, y),$$

and the length of y (the "witness" that shows that $x \in L$) is at most $p(|x|)$.

2.1.1 Randomized Complexity Classes

2.1.1.1 RP: Randomized Polynomial time

Definition 2.1.2 *The class RP consists of all languages L that have a polynomial-time randomized algorithm A with the following behavior:*

- *If $x \notin L$, then A <u>always</u> rejects x (with probability 1).*
- *If $x \in L$, then A accepts x in L with probability at least $\frac{1}{2}$.*

An algorithm that always runs in polynomial time but possibly return erroneous answers is often called a "Monte Carlo" algorithm. Hence, RP admits Monte Carlo algorithms with one-sided error (where the error is in acceptance). Note that while RP is a class of languages, we may call an algorithm A an "RP algorithm" if it satisfies the conditions in the above definition.

Note that the error rate chosen above to be $\frac{1}{2}$ is arbitrary: we could have chosen it to be any positive constant. Indeed, it can be improved through a simple process called *amplification*. Since the error is one-sided, we can repeat the algorithm t times independently: we reject the string x if any of the t runs reject, and accept x otherwise. Since the runs were independent, we have

$$\Pr[\text{ algorithm makes a mistake } t \text{ times }] \leq \frac{1}{2^t}.$$

Thus, we can repeating the algorithm a polynomial number of times and make the error exponentially small. A famous example of a language in RP is *Primes*, the set of all prime numbers: this was shown by Adelman and Huang (1992), who gave a primality testing algorithm that always rejected composite numbers (i.e., numbers $\notin$ *Primes*), and accepted primes with probability at least 1/2.

2.1.1.2 co-RP: complement of RP

Let the complement of the set L be denoted by $\overline{L}$; i.e., $\overline{L} = \Sigma^* - L$. Then language L is in co-RP iff $\overline{L}$ is in RP. A more intuitive definition is the following:

Definition 2.1.3 *The class co-RP consists of all languages L that have a polynomial-time randomized algorithm A with the following behavior:*

- *If $x \notin L$, then A accepts x in L with probability at most $\frac{1}{2}$.*
- *If $x \in L$, then A always accepts x (with probability 1).*

The language *Primes* is also in co-RP: Gary Miller and Michael Rabin gave a primality test that always accepts prime numbers, but rejects composites with probability at least $\frac{1}{2}$. Again, the number $\frac{1}{2}$ could be replaced by any positive constant, since we can use amplification to reduce the probability of error.

2.1.1.3 ZPP: Zero-error Probabilistic Polynomial time

Definition 2.1.4 *A language L is in ZPP if there is an algorithm A that recognizes L (with no error) and runs in expected poly-time.*

Let us stress again that the worst-case running time of the algorithm may not be polynomial, even though the expected running time is polynomial. Another way to define ZPP is in terms of the classes RP and co$-RP$ which we saw above: indeed, the following theorem holds.

Theorem 2.1.5 $ZPP = RP \cap \text{co}-RP$.

Proof: We first show that $ZPP \subseteq RP \cap \text{co}-RP$. Let $L \in ZPP$: hence there is a zero-error algorithm A for L that runs in polynomial expected-time. Let this expected running time be some value t. We will now use this algorithm A to construct algorithms satisfying the requirements for RP and co$-RP$.

Consider the following RP algorithm A' which takes as input x: it runs the algorithm A on x for at most $2t$ steps. If $A(x)$ halts, then we can report the answer given by it. If $A(x)$ does not halt, then we simply reject x. Note that this new algorithm A' always rejects strings not in L. Moreover, if $x \in L$, then A' would reject x only if the computation $A(x)$ ran for more than $2t$ steps. However, since the expected run time for A was only t, the probability that A ran for more than $2t$ steps is at most a half. (Why? This is just *Markov's inequality.*) Thus A' rejects strings in L with probability at most $\frac{1}{2}$: this satisfies the requirements for RP.

To construct an algorithm for co$-RP$, we can do the same thing, but *accept* the input if A does not halt after $2t$ steps. An identical argument shows that the resulting algorithm indeed satisfies the requirements for co-RP. Showing $ZPP \supseteq RP \cap \text{co}-RP$ is left as an exercise for the reader. ■

Since $Primes \in RP$ and $Primes \in \text{co}-RP$, we can conclude that $PRIMES \in ZPP$. It is not known whether $ZPP = P$ or not, but in this case it was recently shown that $Primes \in P$.

Exercise 2.1.6 *Show that* $RP \subseteq NP$ *and co-*$RP \subseteq co - NP$.

2.1.1.4 BPP: Bounded-error Probabilistic Polynomial time

Another complexity class that arises often is the class BPP.

Definition 2.1.7 *The class* BPP *consists of all languages* L *that have a (worst-case) polynomial-time randomized algorithm* A *with the following behavior:*

- *If* $x \in L$, *then* A *accepts* x *in* L *with probability* $\geq \frac{3}{4}$.
- *If* $x \notin L$, *then* A *accepts* x *in* L *with probability* $\leq \frac{1}{4}$.

Hence, the error probability in either case is at most $1/4$.

Again, we can use amplification to decrease the error.

Exercise 2.1.8 *Let* A *be a* BPP *algorithm that accepts* $x \in L$ *with probability at least* $\frac{3}{4}$, *and accepts* $x \notin L$ *with probability at most* $\frac{1}{4}$. *Show that if we run the algorithm* A *for* t *independent iterations and return the majority answer, then the error probability becomes at most* $2^{-O(t)}$.

Note that both RP and co-RP are subsets of BPP. An important open question in complexity theory is whether $BPP \subseteq NP$ or not.

2.2 More Randomized Algorithms

2.2.1 Communication complexity

Alice and Bob each have n-bit numbers a and b, respectively. They want to test whether $a = b$ or not, but there is a communication cost. In particular, they have to pay for each bit Alice sends to Bob or vice versa. The problem is to test for equality while minimizing the cost.

Note that any deterministic algorithm must use n-bits of communication[1]. We now give a randomized protocol that uses $O(\log n)$ bits of communication and succeeds with high probability.

[1]Why? To begin with, Alice has no information about Bob's string b, except that it has one of 2^n values. Each

Here is the protocol:

- Alice picks a prime $p \in [2, \dots, x]$ uniformly at random (we will define x later).
- Alice sends the tuple $\langle p, a \bmod p \rangle$ to Bob.
- Bob computes $b \bmod p$.
- If $a \bmod p = b \bmod p$, then Bob says $a = b$, else he says $A \neq b$.

Clearly, if $a = b$, then Bob can never say $a \neq b$: hence, a mistake is only made if $a \neq b$ and we choose p such that $a \bmod p = b \bmod p$.

Claim 2.2.1 *Set $x = 4n^2$. If $a \neq b$, then $Pr[a \bmod p = b \bmod p] < \frac{1}{2}$.*

Proof: We have $\Pr[\text{failure}] = \Pr[p \text{ divides } (a-b)]$. There can be at most n primes that divide $(a-b)$ (since $|a-b| \leq 2^n$ and every prime number is at least 2). Also, the number of primes in $[2, \dots, x]$ is about $\frac{x}{\log x}$ (Prime Number Theorem); setting $x = 4n^2$ makes this quantity $2n\frac{n}{\log 4n}$, which implies that there are at least $2n$ primes in that interval. So we have:

$$\begin{aligned} \Pr[\text{failure}] &= \frac{\#\text{ of primes that divide } (a-b)}{\#\text{ of primes in } [2, \dots, x]} \\ &\leq \frac{n}{2n} = \frac{1}{2} \end{aligned}$$

■

Moreover, since $x = 4n^2$, both the numbers p and $a \bmod p$ require only $O(\log n)$ bits to represent. Hence the communication required is at most $O(\log n)$ bits, as claimed. (This material is covered in Section 7.4 of M&R.)

2.2.2 Arithmetic circuit checking

Given a circuit of $+$ and $*$ gates with integer inputs and outputs, the arithmetic circuit checking problem is the problem of checking whether the output was computed correctly or not. Since we can get very large numbers which can cancel out later, the goal is to determine whether the answer is correct without recomputing the output completely. A randomized method for solving this problem is similar to the above communication complexity problem, and involves carrying out the algebraic computations mod p.

2.2.3 Matrix multiplication checking

The matrix multiplication checking problem is to verify the process of matrix multiplication. Given three $n \times n$ matrices A, B, and C, is it the case that $AB = C$? The fastest known deterministic algorithm, due to Coppersmith and Winograd, runs in $O(n^{2.376})$ time. Note that an easy lower bound on the running time of any randomized algorithm for this problem is $\Omega(n^2)$ since the input

bit of information partitions this space of 2^n into two halves. After i bits of information, the partition containing a has 2^{n-i} strings. As long as $i < n$, Alice cannot distinguish between these 2^{n-i} strings, and so cannot tell if Bob's string is a or not.

has to at least be read (see Lecture 4 for more details on this). We will now give a randomized algorithm (in co-RP) which takes only $O(n^2)$ time.

Let us introduce some notation for the rest of the course: $x \in_R X$ means "choose x uniformly at random from the set X".

The algorithm is as follows:

- Pick a vector $x \in_R \{0,1\}^n$.
- Compare ABx with Cx (note this takes $O(n^2)$ time since we can compute in the order $A(Bx)$).
- If $ABx = Cx$, then output *Yes*, otherwise output *No*.

Note that if the matrices are over the field $\mathcal{F}$, then the computations are also carried out over the field $\mathcal{F}$. Now if $AB = C$, the algorithm always outputs the correct answer; only if $AB \neq C$, the algorithm may output the wrong answer. We now bound the probability of such an error.

Theorem 2.2.2 (Freivald) *If $AB \neq C$, then the above algorithm fails with probability at most $\frac{1}{2}$.*

Proof: We first prove a simpler claim:

Claim 2.2.3 *Given n-digit strings $a, b \in \mathbb{R}^n$ and $x \in_R \{0,1\}^n$, $Pr[a \cdot x = b \cdot x] \leq \frac{1}{2}$.*

Proof: [Claim] Suppose $a_i \neq b_i$. Let $\alpha = \sum_{j \neq i} a_j x_j$ and $\beta = \sum_{j \neq i} b_j x_j$. We can write $a \cdot x = \alpha + a_i x_i$ and $b \cdot x = \beta + b_i x_i$. This gives us

$$a \cdot x - b \cdot x = (\alpha - \beta) + (a_i - b_i)x_i.$$

We can invoke the Principle of Deferred Decisions (see Section 3.5 of M&R) to assume that we are given x_j for $j \neq i$. Then we can write

$$Pr[a \cdot x - b \cdot x = 0] = Pr\left[x_i = \frac{\alpha - \beta}{b_i - a_i}\right] \leq \frac{1}{2}.$$

■

We now use this claim to prove the theorem: if $AB \neq C$, then there is at least one row in AB, say $(AB)_i$, that differs from the corresponding row C_i in C. Now we apply the above claim with $a = (AB)_i$ and $b = C_i$. The probability that $a \cdot x = b \cdot x$ is at most 1/2, but in order for the algorithm to output *Yes*, we must have $a \cdot x = b \cdot x$. Therefore, the probability of failure for the algorithm is at most 1/2. ■

(See Section 7.1 in M&R for more on this problem.)

2.2.4 Polynomial identity checking

In the polynomial identiy checking problem, we are given two multi-variate polynomials $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ each with degree d (again we are computing in some field $\mathcal{F}$). We are not given the polynomials explicity (we cannot read the polynomials in poly-time). Instead, we have "black-box" access for evaluating a polynomial (for example, see the definition of a *Vandermonde matrix* on page 165 of M&R). Given these two polynomials, the problems is to determine if the polynomials are equal (i.e. $f = g$ or $f - g = 0$). Letting $Q = f - g$, we can check if $Q = 0$.

There is no known poly-time algorithm for this problem. We can show that it is in co-RP.

First consider the univariate case. We can pick $d+1$ distinct, arbitrary values at random from $\mathcal{F}$. If $Q(x) = 0$ for all $d+1$ values for x, then $Q = 0$. (With degree d it can have at most d roots.)

This approach does not directly apply to the multivariate case because there can be exponentially many roots. Roughly speaking, we can handle the multivariate case by fixing $n-1$ variables and applying the result from the univariate case. Consider the following algorithm, which assumes we have some subset $S \subset \mathcal{F}$ with $|S| \geq 2d$.

- Pick $r_1, \ldots, r_n \in_R S$
- Evaluate $Q(r_1, \ldots, r_n)$
- If 0, return $Q = 0$

Theorem 2.2.4 (Schwartz-Zippel) *If, in the above algorithm, $Q \neq 0$, we have*

$$Pr[Q(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Proof: By induction on n. The base case is the univariate case described above. With $Q \neq 0$, we want to compute $Pr[Q(r_1, \ldots, r_n) = 0]$. Let k be the largest power of x_1. We can rewrite

$$Q(x_1, \ldots, x_n) = x_1^k Q_1(x_2, \ldots, x_n) + Q_2(x_1, \ldots, x_n)$$

for some polynomials Q_1 and Q_2. Now we consider two events. Let A be the event that $Q(r_1, \cdots, r_n)$ is 0, and B be the event that $Q_1(r_2, \cdots, r_n)$ is 0.

We can rewrite the probability that $Q_1(r_2, \cdots, r_n)$ is 0 as:

$$\begin{aligned}\Pr[Q(r) = 0] = \Pr[A] &= \Pr[A \mid B]\Pr[B] + \Pr[A \mid \neg B]\Pr[\neg B] \\ &\leq \Pr[B] + \Pr[A \mid \neg B]\end{aligned}$$

Let us first bound the probability of B, or the probability that $Q_1(r_2, \cdots, r_n) = 0$. Q_1 has degree $d-k$ and so we can use the inductive hypothesis to obtain

$$\Pr[B] = \Pr[Q(r_2, \ldots, r_n) = 0] \leq \frac{d-k}{|S|}$$

.

Similarly, given $\neg B$ (or $Q_1(r_2, \cdots, r_n) \neq 0$), the univariate polynomial $Q(x_1, r_2, \ldots, r_n)$ has degree k. Therefore, again by inductive hypothesis,

$$\Pr[A \mid \neg B] = \Pr[Q(x_1, r_2, \ldots, r_n) = 0 \mid Q_1(r_2, \ldots, r_n) \neq 0] \leq \frac{k}{|S|}.$$

Thus,

$$
\begin{aligned}
Pr[Q(r)=0] &\leq Pr[B] + Pr[A \mid \neg B] \\
&\leq \frac{d-k}{|S|} + \frac{k}{|S|} \\
&= \frac{d}{|S|}
\end{aligned}
$$

■

As mentioned before, there is no known poly-time algorithm for this problem. Recent results by Impagliazzo and Kabanets (2003) show that proving that p.i.c. is in P would imply that either $NEXP$ cannot have poly-size non-uniform circuits, or *Permanent* cannot have poly-size non-uniform circuits. Little is known about these lower bound results, and the Impagliazzo-Kabanets result indicates that proving p.i.c. $\in$ P may require new techniques in complexity theory.

(For more on the polynomial identity checking problem, see Section 7.2 in M&R.)

References

L. Adleman and M-D A. Huang (1992). Primality Testing And Two Dimensional Abelian Varieties Over Finite Fields. In *Springer Verlag Lecture Notes In Mathematics*, 1992.

Impagliazzo and Kabanets (2003). Derandomizing polynomial identity checking means proving circuit lower bounds. In *ACM Symposium on the Theory of Computation*, 2003.