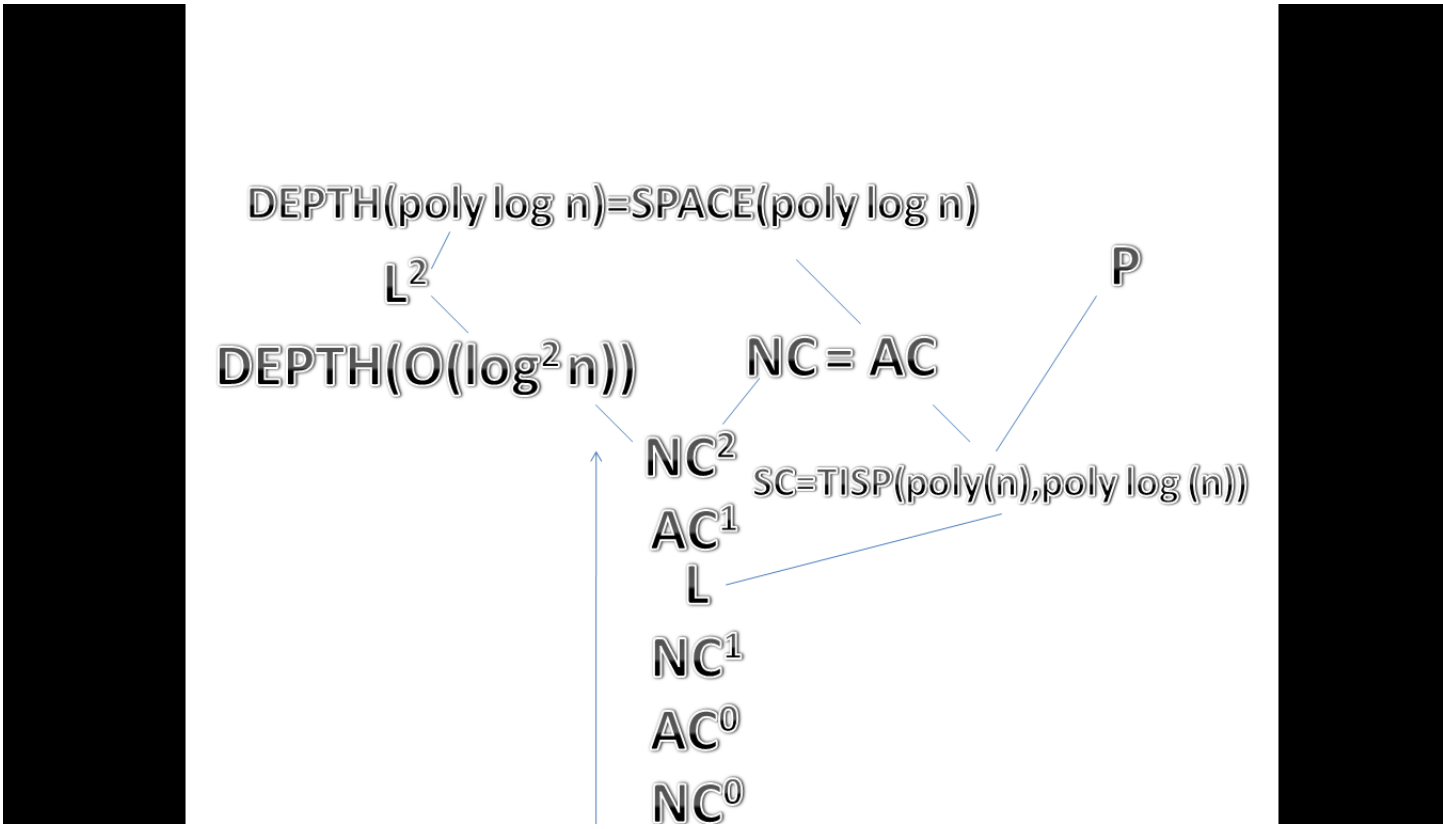
CS 221: Computational Complexity	Prof. Salil Vadhan
Lecture Notes 11	
March 5, 2010	Scribe: Michael Jemison

1 Circuit Complexity Diagram



2 Randomized Computation

Now we'll study the complexity theory of randomized algorithms — algorithms that may make random choices or "toss coins." There are many equivalent formalizations of this idea:

1. Coin-tossing Oracle: When the algorithm goes into a special coin-tossing state, it received a randomly chosen bit (0 or 1 with equal probability) on its current tape location.

2. Randomized Transition Function: The algorithm has two transition functions δ_0 and δ_1, and one of the two randomly chosen (with equal probability) and applied at each step of the computation.

3. Random Tape: The algorithm has a special tape that is filled with an infinite sequence of random bits at the beginning of the computation. However, for space-bounded computations, we

must be careful to restrict the algorithm to have one-way access to the special random tape (as we do not want to give it free storage of its random bits).

3 Polynomial Identity Testing

We begin with an example of a randomized algorithm, one that easily and efficiently solves a problem for which we do not know any efficient deterministic algorithms.

An *arithmetic formula* is a like a boolean formula but the operations are addition and multiplication with constants given over $\mathbb{Z}$. An example of an arithmetic formula $F(x_1, x_2, x_3, x_4)$ in 4 variables is $F(x_1, x_2, x_3, x_4) = (4x_1 - 2x_2) + (x_3)(x_1) + (x_1 + x_2)(x_3 + x_4)$.

There is a commonly occurring problem of deciding when 2 given arithmetic formulas P and Q compute the same polynomial.[1] If we take $R = P - Q$, this problem reduces to the problem of determining if the arithmetic formula R computes the zero polynomial. This computational problem can be described as a language:

$$\text{AFIT}_{\mathbb{Z}} = \{R : R \text{ is an arithmetic formula over } \mathbb{Z} \text{ that computes the zero polynomial}\}.$$

There are no known sub-exponential time deterministic algorithms for $\text{AFIT}_{\mathbb{Z}}$. However, there is a polynomial-time probabilistic algorithm:

Randomized Algorithm for $\text{AFIT}_{\mathbb{Z}}$: On input $F(x_1, ..., x_n)$:
1. Pick $\alpha_1, ..., \alpha_n \in \mathbb{R}$ uniformly at random from $[1, K]$ where K will be determined later.
2. Evaluate $F(\alpha_1, ..., \alpha_n)$. If $F(\alpha_1, ..., \alpha_n) = 0$ accept, else reject.

Analysis of the Algorithm: If F is indeed identical to the zero function, then the algorithm will always reject. If F is not the zero function, then the algorithm will accept with probability with probability $\leq 1/2$ provided that K is chosen correctly. We may reduce the probability of a false positive to $\leq 2^{-l}$ with l repetitions of the algorithm.

The fact that we can choose K follows from:

Lemma 1 (Schwartz-Zippel Lemma) *Let $p(x_1, ..., x_m)$ be a nonzero polynomial of (total) degree[2] at most d over a field $\mathbb{F}$.[3] If $a_1, a_2, ..., a_m$ are randomly chosen with replacement from $S \subseteq \mathbb{F}$, then* $\Pr[p(a_1, ..., a_m) = 0] \leq d/|S|$.

[1] We say two polynomials are *equal* if they have the same coefficient for every monomial (after expanding and collecting like terms). Over infinite fields or domains (like $\mathbb{Z}$), this is is equivalent to saying that P and Q compute the same function. (This follows from the Schwartz–Zippel Lemma below.) But over finite fields, two distinct polynomials may compute the same function; e.g. x^2 and x compute the same function over $\mathbb{Z}_2$ ($\{0, 1\}$ with arithmetic modulo 2).

[2] The *(total) degree* of a monomial $x_1^{e_1} x_2^{e_2} ... x_m^{e_m}$ is equal to $e_1 + ... + e_m$. The *(total) degree* of a polynomial is the largest (total) degree of its monomials.

[3] A *field* is a set with two operations + and × satisfying several familiar properties that we won't list here, but notably requiring that all nonzero elements have multiplicative inverses. Examples include $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{C}$, and $\mathbb{Z}_p$ ($\{0,1,\ldots,p\text{-}1\}$ with arithmetic modulo a *prime* p). $\mathbb{Z}$ is not a field, but we can apply the Schwartz– Zippel lemma to it because it is contained in the field $\mathbb{Q}$. In contrast, $\mathbb{Z}_n$ for composite n is not a field, nor is it contained in any field.

Proof: The proof proceeds by induction. The case $m = 1$ is clear from the fact that any non-zero polynomial univariate of degree d (over any field) has at most d roots. Suppose the assertion holds for $m = n - 1$. We may extend the assertion to m=n by writing the polynomial $p(x_1, ..., x_n)$ as a univariate polynomial in x_1 with coefficients that are polynomials in $x_2, ..., x_n$. So we would write $p(x) = \sum_i x_1^i p_i(x_2, ..., x_n)$. The full details of the proof are given in Arora Barak Appendix A. ■

The following result will be useful:

Theorem 2 *If $F(x_1, ..., x_n)$ is an arithmetic formula, then $\deg(F) \leq |F|$, where $|F|$ is the size of the formula F (i.e. the number of nodes when F is written as a tree).*

Proof: The proof again proceeds by induction. We induct on $|F|$. Suppose that $|F| = 1$. Then $F = x_i$ or $F = c \in \mathbb{F}$ for some field $\mathbb{F}$. Now suppose that the cases for $|F| \leq n - 1$ have all been proven. Now consider the case $|F| = n$. We have that $F = G + H$ or that $F = G * H$ for $|G| < n$ and $|H| < n$. In either case, we have

$$\begin{aligned} \deg(F) &\leq \deg(G) + \deg(H) \\ &\leq |G| + |H| \\ &\leq |F|. \end{aligned}$$

Where the second inequality follows from the inductive hypothesis. ■

Combining these two results we see that we may choose $K = 2|F|$. So in summary, if $F \neq 0$, the algorithm will accept with probability $\leq \frac{1}{2}$ for $K = 2|F|$.

A variant of the above algorithm also works for arithmetics *circuits* (not just formulas). One issue is that the degree of the polynomial computed by a given circuit C can be exponential in the size of $|C|$. This requires that we take K to be exponentially large (e.g. $K = 2^{|C|+1}$) to apply the Schwartz–Zippel Lemma. This is not a problem, as the numbers $\alpha_1, \ldots, \alpha_n \in \{1, \ldots, K\}$ have binary representations that are polynomially long. However, a bigger problem is that the the intermediate and final values of $F(\alpha_1, \ldots, \alpha_n)$ can have exponentially large bitlength. (A similar proof as the above shows that this does not happen with formulas.) This can be solved by doing the entire computation modulo a random prime number of polynomial bitlength; see Arora–Barak for details.

4 Probabilistic Language Classes

The main complexity class associated with efficient randomized algorithms is the following:

Definition 3 *A language $L \in$ **BPP** ("bounded-error probabilistic polynomial time") if there is a probabilistic poly-time algorithm A (i.e. A always halts in poly-time) such that $x \in L \Rightarrow \Pr[A(x) = 1] \geq \frac{2}{3}$ and $x \notin L \Rightarrow \Pr[A(x) = 1] \leq \frac{1}{3}$.*

Sometimes it is useful to focus on algorithms that have one-sided error:

Definition 4 *The definition for* **RP** *(randomized polynomial time) is similar however we have the differences that* $L \in \mathbf{RP}$ *if* $x \in L \Rightarrow \Pr[A(x) = 1] \geq \frac{1}{2}$ *and* $x \notin L \Rightarrow \Pr[A(x) = 1] = 0$.

As usual, **co-RP** *is defined to consists of the complements of languages in* **RP**. *Equivalently,* $L \in$ **co-RP** *if there is a probabilistic poly-time* A *such that* $x \in L \Rightarrow \Pr[A(x) = 1] = 1$ *and* $x \notin L \Rightarrow \Pr[A(x) = 1] \leq \frac{1}{2}$.

It follows from the algorithm of the previous section that:

Theorem 5 $\mathrm{AFIT}_{\mathbb{Z}} \in$ **co-RP**.

For **RP** (**co-RP**) we can reduce the error to 2^{-k} by doing k repetitions and accepting if at least one (all) of the repetitions accept.

Theorem 6 *If we repeat a* **BPP** *algorithm* $l = \mathrm{poly}(n)$ *times and rule by majority vote, then the error probability will be at* $2^{-\Omega(l)}$, *where the hidden constant depends on the initial constant gap in the acceptance probability of the* **BPP** *algorithm on yes and no instances.*

Proof: We use the **Chernoff Bound:** Let $X_1, ..., X_l$ be independent $[0, 1]$-valued random variables, $X = \frac{1}{l}\Sigma_i X_i$ their average, and $\mu = \mathrm{E}[X]$ the expectation of X. Then for every $\epsilon > 0$,

$$\Pr[|X - \mu| \geq \epsilon] \leq 2^{-\Omega(\epsilon^2 l)}.$$

To apply this to the **BPP** error reduction we define $X_i = 1$ if the algorithm is correct in the i'th repetition (on a fixed input) and $X_i = 0$ otherwise. Then

$$E[X_i] \geq \frac{2}{3} \Rightarrow \mu \geq \frac{2}{3}$$

$$\Pr[X \leq 1/2] \leq \Pr[|X - \mu| \geq \frac{1}{6}] = 2^{-\Omega(\frac{1}{6}^2 l)} = 2^{-\Omega(l)}$$

■

Notice that, in general, the rate at which the error decreases depends quadratically on the (half of) the gap between the algorithm's acceptance probability on inputs in the language and inputs not in the language (which is $1/6$ in the standard definition of **BPP**). Indeed, this error reduction does not work if we do not require this gap to be at least $1/\mathrm{poly}(n)$. In particular, it does not work for the following class:

Definition 7 *A language* $L \in$ **PP** *if there is a probabilistic poly-time algorithm* A *such that* $x \in L \Rightarrow \Pr[A(x) = 1] > \frac{1}{2}$ *and* $x \notin L \Rightarrow \Pr[A(x) = 1] \leq \frac{1}{2}$.

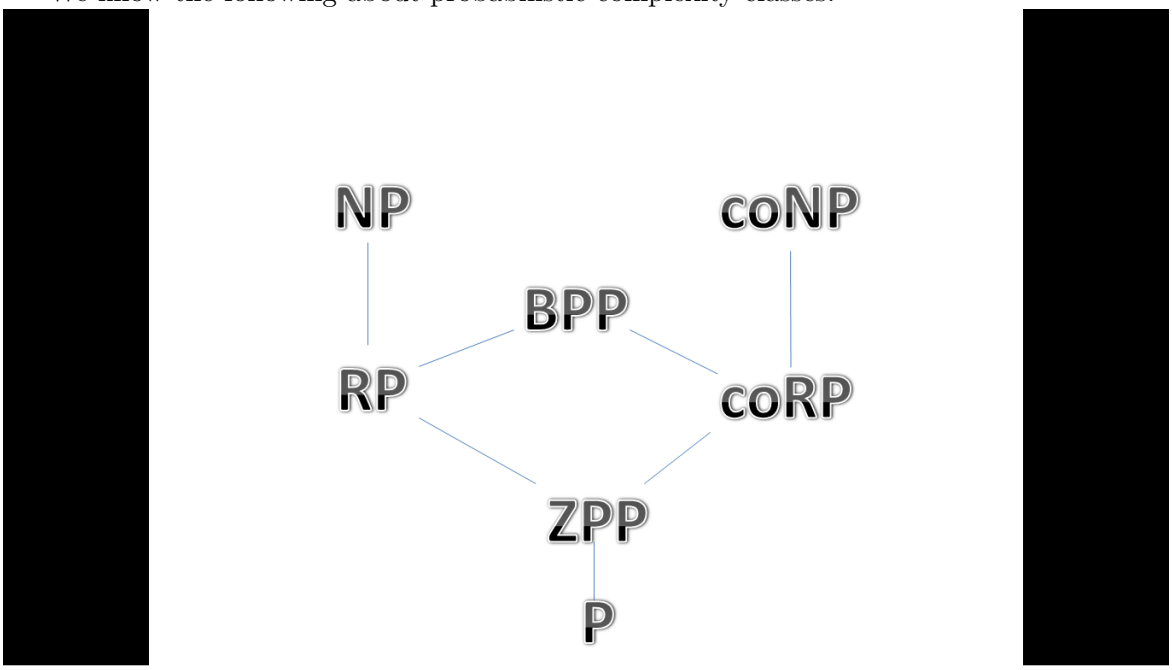
Even though **PP** stands for "probabilistic polynomial time," it is not a reasonable model for randomized algorithms, as gap in acceptance probability between inputs in the language and not in the language can be exponentially small.

A similar phenomenon occurs for one-sided error. If we eliminate the gap for **RP** and just require that $x \in L \Rightarrow \Pr[A(x) = 1] > 0$, then we obtain the class **NP** — with the randomization being better thought of as nondeterminism.

Definition 8 ZPP *is the set of languages decidable by probabilistic algorithms that never err but run in expected poly-time, i.e. there is a polynomial p, such that for all x, we have* $\mathrm{E}[Time(A(x))] \leq p(|x|)$.

Fact: $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{co\text{-}RP}$.

We know the following about probabilistic complexity classes:



Next time: $\mathbf{BPP} \subseteq \mathbf{\Sigma_2^p} \cap \mathbf{\Pi_2^p}$.

Examples of Problems where randomized algorithms are exponentially faster than known deterministic ones:

1. Polynomial Identity Testing (as above).
1. Approximate counting problems such as the number of perfect matchings in a graph.
2. Estimating partition functions in statistical physics.
3. Extracting square roots mod a prime.

Thus, randomization *appears* to be very powerful. However, there is also strong evidence that actually every randomized algorithm can be made deterministic with only a polynomial slowdown:

Theorem 9 (Impagliazzo–Wigderson) *If* SAT *(or any problem in* $\mathbf{DTIME}(2^{O(n)})$*) requires exponentially large circuits, i.e.* $2^{\Omega(n)}$*, then* $\mathbf{BPP} = \mathbf{P}$.

We will not cover this result in CS221, but if you are interested, see the later chapters of Arora–Barak or take CS225 ("Pseudorandomness", to be offered in Spring 2011).