Notes on Complexity Theory Last updated: October, 2011

Lecture 12

Jonathan Katz

1 Randomized Time Complexity

Is deterministic polynomial-time computation the only way to define "feasible" computation? Allowing *probabilistic* algorithms, that may fail with tiny probability, seems reasonable. (In particular, consider an algorithm whose error probability is lower than the probability that there will be a hardware error during the computation, or the probability that the computer will be hit by a meteor during the computation.) This motivates our exploration of probabilistic complexity classes.

There are two different ways to define a randomized model of computation. The first is via Turing machines with a *probabilistic transition function*: as in the case of non-deterministic machines, we have a Turing machine with two transitions functions, and a *random* one is applied at each step. The second way to model randomized computation is by augmenting Turing machines with an additional (read-only) *random tape*. For the latter approach, one can consider either one-way or two-way random tapes; the difference between these models is unimportant for randomized time complexity classes, but (as we will see) becomes important for randomized space classes. Whichever approach we take, we denote by $M(x)$ a random computation of M on input x, and by $M(x;r)$ the (deterministic) computation of M on input x using random choices r (where, in the first case, the ith bit of r determines which transition function is used at the ith step, and in the second case r is the value written on M's random tape).

We now define some randomized time-complexity classes; in the following, PPT stands for "probabilistic, polynomial time" (where this is measured as *worst-case* time complexity over all inputs, and as always the running time is measured as a function of the length of the input x).

Definition 1 *$L \in \mathcal{RP}$ if there exists a* PPT *machine M such that:*

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = 1] \geq 1/2 \\ x \notin L &\Rightarrow \Pr[M(x) = 0] = 1. \end{aligned}$$

Note that if $M(x)$ outputs "1" we are sure that $x \in L$; if $M(x)$ outputs "0" we cannot make any definitive claim.

Viewing M as a non-deterministic machine for L, the above means that when $x \in L$ at least half of the computation paths of $M(x)$ accept, and when $x \notin L$ then none of the computation paths of $M(x)$ accept. Put differently, a random tape r for which $M(x;r) = 1$ serves as a witness that $x \in L$. We thus have $\mathcal{RP} \subseteq \mathcal{NP}$.

Symmetrically:

Definition 2 *$L \in \mathsf{co}\mathcal{RP}$ if there exists a* PPT *machine M such that:*

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = 1] = 1 \\ x \notin L &\Rightarrow \Pr[M(x) = 0] \geq 1/2. \end{aligned}$$

Here, if $M(x)$ outputs "0' we are sure that $x \notin L$, but if $M(x)$ outputs "1" we cannot make any definitive claim.

The above classes allow *one-sided* error. A more general notion of randomized computation allows for *two-sided* error. For a language L, let $\chi_L(x) = 1$ iff $x \in L$.

Definition 3 *$L \in \mathcal{BPP}$ if there exists a* PPT *machine M such that:*

$$\Pr[M(x) = \chi_L(x)] \geq 2/3.$$

In other words,

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = 1] \geq 2/3 \\ x \notin L &\Rightarrow \Pr[M(x) = 1] \leq 1/3. \end{aligned}$$

Finally, we may also consider randomized algorithms that make no errors (but may not give a result at all):

Definition 4 *$L \in \mathcal{ZPP}$ if there exists a* PPT *machine M such that:*

$$\Pr[M(x) = \chi_L(x)] \geq 1/2$$

$$\Pr\left[M(x) \in \{\chi_L(x), \perp\}\right] = 1.$$

We now explore these definitions further. A first observation is that, for all the above definitions, the constants are essentially arbitrary. We focus on the case of $\mathcal{BPP}$ and leave consideration of the rest as an exercise.

Theorem 1 *The following are both equivalent definitions of $\mathcal{BPP}$:*

1. *$L \in \mathcal{BPP}$ if there exists a* PPT *machine M and a polynomial p such that:*

$$\Pr[M(x) = \chi_L(x)] \geq \frac{1}{2} + \frac{1}{p(|x|)}.$$

2. *$L \in \mathcal{BPP}$ if there exists a* PPT *machine M and a polynomial q such that:*

$$\Pr[M(x) = \chi_L(x)] \geq 1 - 2^{-q(|x|)}.$$

Proof We show how to transform an algorithm M satisfying the first definition into an algorithm M' satisfying the second definition. $M'(x)$ is defined as follows: run $M(x)$ a total of $t(|x|)$ times (for some polynomial t to be fixed later) using independent random coins in each execution. Then M' outputs the bit that was output by a majority of these executions.

To analyze the behavior of M', we rely on the *Chernoff bound* [**?**, Chap. 4]:

Claim 2 *Let $p \leq \frac{1}{2}$ and let $X_1, \ldots, X_n$ be independent, identically-distributed 0-1 random variables with $\Pr[X_i = 1] = p$ for each i. Then for all ε with $0 < \varepsilon \leq p(1-p)$ we have:*

$$\Pr\left[\left|\frac{\sum_{i=1}^n X_i}{n} - p\right| > \varepsilon\right] < 2 \cdot e^{-\frac{\varepsilon^2 n}{2p(1-p)}} \leq 2 \cdot e^{-2n\varepsilon^2}.$$

Let X_i denote the output of the i^{th} execution of $M(x)$. When $x \notin L$

$$\Pr[X_i = 1] < \frac{1}{2} - \frac{1}{p(|x|)} \stackrel{\text{def}}{=} \rho.$$

Furthermore, by definition of M' (letting $t \stackrel{\text{def}}{=} t(|x|)$):

$$\begin{aligned} \Pr[M'(x) = 1] &= \Pr\left[\frac{\sum_{i=1}^{t} X_i}{t} > \frac{1}{2}\right] \\ &\leq \Pr\left[\left|\frac{\sum_{i=1}^{t} X_i}{t} - \rho\right| > \frac{1}{p(|x|)}\right] \\ &< 2 \cdot e^{-\frac{2t}{p(|x|)^2}}. \end{aligned}$$

Setting $t = O\left(q(|x|) \cdot p(|x|)^2\right)$ gives the desired result. (An exactly analogous argument works for the case $x \in L$.) ■

How do the above classes relate to each other? It is immediate that

$$\mathcal{RP} \cup \mathsf{co}\mathcal{RP} \subseteq \mathcal{BPP},$$

and so $\mathcal{BPP}$ is a (potentially) more powerful class. Indeed, $\mathcal{BPP}$ appears to capture feasible probabilistic computation. We also claim that

$$\mathcal{ZPP} = \mathcal{RP} \cap \mathsf{co}\mathcal{RP};$$

this is left as an exercise. A third characterization of $\mathcal{ZPP}$ is in terms of expected polynomial-time algorithms that always output the correct answer. Let M be a probabilistic Turing machine. We say that M runs in expected time $t(n)$ if, for every $x \in \{0,1\}^n$, the expected running time of $M(x)$ is at most $t(n)$. Then:

Claim 3 *$L \in \mathcal{ZPP}$ iff there exists an expected polynomial-time Turing machine M such that*

$$\Pr[M(x) = \chi_L(x)] = 1.$$

How about a "minimal" notion of correctness for probabilistic algorithms, where we only require correctness with probability arbitrarily better than guessing? This gives rise to a class called $\mathcal{PP}$:

Definition 5 *$L \in \mathcal{PP}$ if there exists a* PPT *machine M such that:*

$$\Pr[M(x) = \chi_L(x)] > 1/2.$$

In other words,

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = 1] > 1/2 \\ x \notin L &\Rightarrow \Pr[M(x) = 1] < 1/2. \end{aligned}$$

A little thought shows that this is *not* a reasonable notion of probabilistic computation. The problem is that the gap between outputting the correct answer and the wrong answer might be exponentially small (in contrast to $\mathcal{BPP}$, where the gap must be some inverse polynomial); in particular, amplification does not work here. As some further evidence against the reasonableness of $\mathcal{PP}$, we have $\mathcal{NP} \subseteq \mathcal{PP}$ (this, too, is left as an exercise); thus, this notion of probabilistic computation can solve all of $\mathcal{NP}$!

1.1 Examples of Randomized Algorithms

There are several examples of where randomized algorithms are more efficient, or simpler, than known deterministic algorithms. However, there are not as many examples of problems that are known to be solvable by polynomial-time randomized algorithms, but not known to be solved by polynomial-time deterministic algorithms. One famous former example was testing primality: this problem was known to be in $\mathsf{co}\mathcal{RP}$ since the late 1970s, but was only shown to be in $\mathcal{P}$ in 2005. (Nevertheless, in practice the randomized algorithms are still used since they are faster.)

A search problem for which probabilistic polynomial-time algorithms are known, but deterministic polynomial-time algorithms are not, is computing square roots modulo a prime.

Polynomial identity testing. Another interesting example is given by the problem of testing equality of *arithmetic* circuits. Here we work with circuits that take integers (rather than boolean values) as input, and where gates compute $+$, $-$, and $\times$ (over the integers); the output is an integer. Say we want to test whether two circuits C_1, C_2 compute the same function. Note that this easily reduces to deciding the following language:

$$\mathsf{ZEROP} \stackrel{\text{def}}{=} \{C \mid C \text{ outputs 0 on all inputs}\}.$$

Any arithmetic circuit is equivalent to a multivariate polynomial over the integers; in principle, then, we can decide membership in ZEROP by expanding and writing out the polynomial to see whether it is identically 0. (This explains the name ZEROP for the language above: we are testing whether an implicitly defined polynomial is the 0 polynomial.) In an arithmetic circuit with m gates, however, the (total) degree[1] of the equivalent polynomial can be as high as 2^m, and so even just writing out all the terms of the polynomial may require exponential time! This is therefore not a viable approach for an efficient algorithm.

In fact, there is no known (efficient) deterministic algorithm for this problem. Instead, we make use of the Schwartz-Zippel lemma (which is useful in many contexts). We state it here for polynomials over the integers, but it also holds over any field.

Lemma 4 *Let $p(X_1, \ldots, X_n)$ be a non-zero polynomial of total degree at most d, and let S be any finite set of integers. Then*

$$\Pr_{x_1,\ldots,x_n \leftarrow S}[p(x_1, \ldots, x_n) = 0] \leq d/|S|.$$

Proof The proof is by induction on n. When $n = 1$, a non-zero polynomial $p(X_1)$ of degree at most d has at most d roots and the lemma follows. Now assume the lemma is true for $n-1$ and prove that it holds for n. Given a polynomial $p(X_1, \ldots, X_n)$ of total degree d, we may write

$$p(X_1, \ldots, X_n) = \sum_{i=0}^{d'} p_i(X_1, \ldots, X_{n-1}) \cdot X_n^i, \tag{1}$$

for some $d' \leq d$ and $p_{d'}(X_1, \ldots, X_{n-1})$ a non-zero polynomial of total degree at most $d - d'$. When $x_1, \ldots, x_{n-1}$ are chosen at random from S, the inductive assumption tells us that $p_{d'}(x_1, \ldots, x_{n-1}) = 0$ with probability at most $(d - d')/|S|$. When $p_{d'}(x_1, \ldots, x_{n-1}) \neq 0$, then (1) is a polynomial of

[1] The total degree of a monomial is the sum of the degrees in each variable; the total degree of a multivariate polynomial largest total degree of any monomial.

degree d' in the single variable X_n, and so the probability (over random choice of $x_n \in S$) that $p(x_1, \ldots, x_{n-1}, x_n) = 0$ is at most $d'/|S|$. Putting everything together, we have

$$\begin{aligned} \Pr_{x_1,\ldots,x_n \leftarrow S}[p(x_1, \ldots, x_n) = 0] &\leq \frac{d-d'}{|S|} + \left(1 - \frac{d-d'}{|S|}\right) \cdot \frac{d'}{|S|} \\ &\leq \frac{d}{|S|}. \end{aligned}$$

This completes the proof. ■

The above suggests a simple randomized algorithm: given an arithmetic circuit C with m gates, and n inputs $X_1, \ldots, X_n$, choose $x_1, \ldots, x_n \leftarrow \{1, \ldots, 2^{m+1}\}$ and evaluate $C(x_1, \ldots, x_n)$. If the output is 0, accept; if the output is non-zero, reject. If $C \in \mathsf{ZEROP}$, then this algorithm always accepts, while if $C \notin \mathsf{ZEROP}$ then the algorithm rejects with probability at least $1/2$. (This is thus a $\mathsf{co}\mathcal{RP}$ algorithm for ZEROP.)

There is, however, a problem with the algorithm: as written, it is not efficient. The difficulty is that the value of $C(2^{m+1}, \ldots, 2^{m+1})$ may be as high as $(2^{m+1})^{2^m}$, which would require exponentially many bits to write down. We can solve this by "fingerprinting"; see [1] for details.

Perfect matching in bipartite graphs. We can use similar ideas to give an efficient randomized algorithm for detecting the existence of a perfect matching in a bipartite graph. (Although this problem is in $\mathcal{P}$, the randomized algorithm we show can be implemented in randomized-NC.) Let G be an $n \times n$ matrix representing a bipartite graph on $2n$ vertices, where $G_{i,j} = X_{i,j}$ if there is an edge from i to j, and $G_{i,j} = 0$ otherwise. The determinant of G is

$$\det(G) = \sum_{\sigma} (-1)^{\mathrm{sign}(\sigma)} \prod_{i=1}^{n} G_{i,\sigma(i)},$$

and we see that $\det(G)$ is a non-zero polynomial (in the variables $X_{1,1}, \ldots, X_{n,n}$) iff the underlying graph has a perfect matching. Calculating the polynomial $\det(G)$ cannot be done efficiently, since it may have exponentially many terms; however, we can evaluate $\det(G)$ for any given values o using standard algorithms for computing the determinant. We can thus use the Shwartz-Zippel lemma and the ideas seen previously to construct a randomized algorithm for this problem.

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.