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 transition 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 \( i \)th bit of \( r \) determines which transition function is used at the \( i \)th 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:

\[
\begin{align*}
  x \in L & \implies \Pr[M(x) = 1] \geq 1/2 \\
  x \notin L & \implies \Pr[M(x) = 0] = 1.
\end{align*}
\]

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.

We thus have \( \mathcal{RP} \subseteq \mathcal{NP} \).

**Symmetrically:**

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

\[
\begin{align*}
  x \in L & \implies \Pr[M(x) = 1] = 1 \\
  x \notin L & \implies \Pr[M(x) = 0] \geq 1/2.
\end{align*}
\]
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:

$$\Pr[M(x) = \chi_L(x)] \geq \frac{2}{3}.$$  

In other words,

$$x \in L \implies \Pr[M(x) = 1] \geq \frac{2}{3}$$

$$x \notin L \implies \Pr[M(x) = 1] \leq \frac{1}{3}.$$

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:

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

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

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 additive version of the Chernoff bound [?]:

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

$$\Pr\left[\left|\sum_{i=1}^{t} X_i - pt\right| > \varepsilon t\right] < 2e^{-2t\varepsilon^2}$$
Although we don’t use it here, there is also a very useful Chernoff bound on the multiplicative error:

**Claim 3** Let \( \rho \leq \frac{1}{2} \) and let \( X_1, \ldots, X_t \) be independent, identically-distributed 0-1 random variables with \( \Pr[X_i = 1] = \rho \) for each \( i \). Then for all \( \epsilon > 0 \), we have:

\[
\Pr \left[ \sum_{i=1}^{t} X_i > (1 + \epsilon) \rho t \right] \leq e^{-\frac{\rho \epsilon t^2}{3}}
\]

and

\[
\Pr \left[ \sum_{i=1}^{t} X_i < (1 - \epsilon) \rho t \right] \leq e^{-\frac{\rho \epsilon t^2}{2}}
\]

Let \( X_i \) denote the output of the \( i \)th execution of \( M(x) \). When \( x \not\in L \)

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

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

\[
\Pr[M'(x) = 1] = \Pr \left[ \frac{\sum_{i=1}^{t} X_i}{t} > \frac{1}{2} \right] \leq \Pr \left[ \frac{\sum_{i=1}^{t} X_i}{t} > \rho + \frac{1}{p(|x|)} \right] < e^{-\frac{2t}{p(|x|)^2}}
\]

Setting \( t = O (q(|x|) \cdot p(|x|)^2) \) 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

\[
\mathcal{RP} \cup \mathsf{coRP} \subseteq \mathcal{BPP},
\]

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

\[
\mathcal{ZPP} = \mathcal{RP} \cap \mathsf{coRP};
\]

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 4 \( L \in \mathbb{ZPP} \) iff there exists an expected polynomial-time Turing machine \( M \) such that
\[
\Pr[M(x) = \chi_L(x)] = 1.
\]

Proof Let \( M_L \) be the machine that decides \( L \) with probability \( 1/2 \), and aborts with probability \( 1/2 \). Let’s denote by \( t(n) \) the (worst-case) run-time of \( M_L \). Let \( M \) be the machine that runs \( M_L \) repeatedly using independent, uniformly chosen random coins in each execution, until \( M_L \) outputs either 1 or 0. Clearly \( M \) decides \( L \), so we only need to analyze its expected run-time. The expected run-time of \( M \) is defined as \( \sum_{i \in \mathbb{N}} i \cdot \Pr[M \text{ runs for } i \text{ steps }] \). Note that with probability \( 1/2 \), \( M \) executes \( M_L \) only once, with probability \( 1/4 \) it executes it exactly twice, and so forth. On some fixed input of length \( n \), letting \( t = t(n) \) denote the worst-case run-time of \( M_L \) on inputs of that size, we can bound the expected run-time of \( M \) by
\[
\sum_{i \in \mathbb{N}} (t \cdot i) \frac{1}{2^i} = t \sum_{i \in \mathbb{N}} \frac{i}{2^i} = 2t.
\]

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 \( \mathbb{P} \mathbb{P} \):

Definition 5 \( L \in \mathbb{P} \mathbb{P} \) if there exists a ppt machine \( M \) such that:
\[
\Pr[M(x) = \chi_L(x)] > 1/2.
\]

In other words,
\[
x \in L \implies \Pr[M(x) = 1] > 1/2
\]
\[
x \notin L \implies \Pr[M(x) = 1] < 1/2.
\]

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 \( \mathbb{B} \mathbb{P} \mathbb{P} \), where the gap must be some inverse polynomial); in particular, amplification does not work here. As some further evidence against the reasonableness of \( \mathbb{P} \mathbb{P} \), we have \( \mathbb{N} \mathbb{P} \subseteq \mathbb{P} \mathbb{P} \) (this, too, is left as an exercise); thus, this notion of probabilistic computation can solve all of \( \mathbb{N} \mathbb{P} \)!

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 \( \mathbb{C} \mathbb{O} \mathbb{R} \mathbb{P} \) since the late 1970s, but was only shown to be in \( \mathbb{P} \) in 2005. (Nevertheless, in practice the randomized algorithms are still used since they are faster.)

Before going through a few examples, we introduce the concept of a finite field. In our case, we will focus only on fields of prime order (size): for some prime \( p \), the integers \( \{0, 1, \ldots, p - 1\} \) constitute the field \( \mathbb{F}(p) \), where addition and multiplication are done modulo \( p \). For example, in \( \mathbb{F}(7) \), we have \( 5 \times 3 = 1 \), and \( 6 + 4 = 3 \).
Claim 5 *Given elements* $a, b, c \in \mathcal{G} \mathcal{F}(P)$ *such that* $a \cdot b \equiv a \cdot c$, *either* $a \equiv 0$, *or* $b \equiv c$.

**Proof** If we instead perform operations over the integers, our assumptions equivalently state that $a \cdot b = a \cdot c + k \cdot p$ for some integer $k$. It follows that $a(b - c) = k \cdot p$, which implies that either $a \equiv 0 \mod p$, or $(b - c) \equiv 0 \mod p$. ■

Claim 6 *For any* $a, b \in \mathcal{G} \mathcal{F}(p)$ *where* $a$ *is non-zero, and for* $r$ *sampled uniformly from* $\mathcal{G} \mathcal{F}(p)$, $\Pr[a \cdot r = b] = 1/p$.

**Proof** By the Claim ??, and because we know that $a$ is nonzero, there is only one value $r \in \mathcal{G} \mathcal{F}$ for which $a \cdot r \equiv b$. The probability of choosing this value is $1/p$. ■

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. We will instead give a randomized algorithm for a problem that has a simple, but less efficient deterministic algorithm.

**Matrix multiplication testing.** The language we’re interested in is

\[
\{(A, B, C) | AB = C, \text{ and each element is a matrix in } \mathcal{G} \mathcal{F}(p)^{n \times n}\}
\]

A simple deterministic algorithm follows by matrix multiplication: simply compute $AB$ and compare each of the $n^2$ elements against those of $C$. This takes time $O(n^3)$ if the multiplication is done naively. The best known construction for matrix multiplication is $O(n^{2.3729})$, though the algorithm would be quite complex. We give a randomized algorithm that takes time $O(n^2)$, with error $1/p$. Repeating will lower the error rate, and, of course the exact run-time will depend on the desired error. The algorithm is simple: instead of computing $AB$, which is expensive, we sample a random vector $\vec{r} \leftarrow \mathcal{G} \mathcal{F}(p)^n$, and compare $A \cdot (B \cdot r)$ with $C \cdot r$, which takes time $O(n^2)$. We now analyze the error probability. Clearly the error is one-sided: if $AB = C$, then $A \cdot (B \cdot r) \equiv C \cdot r$. So we assume that $AB$ differs from $C$ in at least one place, which we’ll let be $(i, j)$. For the sake of analysis, it is easier to consider $AB - C = D$, and look at the probability that $D \cdot r \equiv 0$. Consider the ith element of $D \cdot r$, which is $d_{i,1} r_1 + d_{i,2} r_2 + \cdots + d_{i,j} r_j + \cdots + d_{i,n} r_n = d_{i,j} r_j + y$ for some element $y \in \mathcal{G} \mathcal{F}(p)$. The probability that this value is 0 can be written as

\[
\Pr[(d_{i,j} r_j + y) = 0 \mid y = 0] \cdot \Pr[y = 0] + \Pr[(d_{i,j} r_j + y) = 0 \mid y \neq 0] \cdot \Pr[y \neq 0].
\]

Because we know that $d_{i,j}$ is nonzero, $\Pr[(d_{i,j} r_j + y) = 0 \mid y = 0] = 1/p$ (i.e. the condition is satisfied only when $r_j = 0$). From Claim ??, we know that $\Pr[(d_{i,j} r_j + y) = 0 \mid y \neq 0] = 1/p$: there is exactly one additive inverse of $y$ (namely, $p - y$), and the probability that $d_{i,j} \cdot r_j = p - y$ is $1/p$. Putting these facts together, we have that the probability that the ith element of $D \cdot r \equiv 0$ is

\[
\frac{1}{p} \cdot \Pr[y = 0] + \frac{1}{p} \cdot \Pr[y \neq 0] = \frac{1}{p}
\]

**References**
