1 Randomized Computation

Why is randomness useful? Imagine you have a stack of bank notes, with very few counterfeit ones. You want to choose a genuine bank note to pay at a store. However, suppose that you don’t know how to distinguish between a “good” bank note and a “bad” one. What can you do? Well, if you pick a bank note at random, you will be lucky with high probability (here the probability of picking a good bank note is equal to the fraction of good bank notes in your stack).

1.1 Communication complexity

Let’s consider a more realistic example. Alice and Bob communicate over some channel. The communication is very expensive. Both Alice and Bob have an $n$-bit number. Alice has $a = a_1 \ldots a_n$ and Bob has $b = b_1 \ldots b_n$. They want to know if $a = b$.

Clearly, this check can be achieved with at most $n + 1$ communicated: Alice just sends her number $a$ to Bob. Bob then compares $a$ and $b$, and sends back to Alice one bit (1 if $a = b$, and 0 otherwise). If we allow only deterministic protocols, then this bound is the best one can get.

However, if we allow randomized protocols, we can do much better. Here, a randomized protocol may have Alice and Bob make a mistake (e.g., they may think that $a = b$ when in fact $a \neq b$), but this error should happen with very small probability over the random choices of Alice and Bob. Next, we’ll describe a $O(\log n)$-communication protocol to check if $a = b$.

Alice and Bob will first find the smallest prime number $p$ such that $n^2 \leq p \leq n^3$. (The high density of primes guarantees that there will be a prime between $n^2$ and $n^3$; in the interval $1, 2, \ldots, m$, there are about $\Theta(m/\ln m)$ primes.) Then Alice picks a random element $r \in Z_p$, computes $A(r) = a_1 r^{n-1} + a_2 r^{n-2} + \cdots + a_n \mod p$, and sends to Bob the pair $(r, A(r))$. Upon receiving this pair, Bob will compute $B(r) = b_1 r^{n-1} + b_2 r^{n-2} + \cdots + b_n \mod p$ and test if $A(r) = B(r)$. If they are equal, Bob will send 1 to Alice (saying that he thinks that $a = b$); otherwise, Bob will send 0 to Alice (saying that he thinks $a \neq b$).

Observe that the amount of communication for the described protocol is at most $2|p| + 1 \in O(\log n)$, by our choice of $p \leq n^3$. Now let’s analyze the correctness. First, if $a = b$, then
$A(x) = B(x)$ as polynomials, and hence, $A(r) = B(r)$ for any $r$. So in this case, Alice and Bob correctly decide that $a = b$ with probability 1.

In the case where $a \neq b$, we have that $A(x)$ and $B(x)$ are different polynomials of degree at most $n - 1$. So, their difference $C(x) = A(x) - B(x)$ is a non-zero polynomial of degree at most $n - 1$. The polynomial $C(x)$ remains nonzero even if we view it as a polynomial over the $Z_p$ (over the finite field consisting of numbers between 0 and $p - 1$, where all arithmetic operations, addition, subtraction, multiplication, and division, are done modulo $p$). This is because the nonzero coefficients of $C(x)$ are 1, $-1$, and so they remain nonzero even modulo $p$.

We will use the following basic fact about polynomials over finite fields: A nonzero polynomial of degree $d$ over a finite field can have at most $d$ roots. This fact should be familiar to you in the case of polynomials over the field of real numbers. It is also true over finite fields.\footnote{When $p$ is not prime, “strange” things happen. For example, the product of two nonzero polynomials may become a zero polynomial: the product of $2x$ and $3x - 3$ yields $6x^2 - 6x = 0 \mod 6$. The source of the “problem” here is the existence of nonzero numbers modulo 6, like 2 and 3, whose product is zero modulo 6. Such a problem never arises for prime $p$.}

The probability that Alice and Bob erroneously decide that $a = b$ is exactly $\Pr_{r \in Z_p}[C(r) = 0]$, where all computations are modulo $p$. Now, since $C(x)$ is of degree at most $n - 1$, then by the aforementioned fact, it may have at most $n - 1$ roots, i.e., values $r$ at which $C$ is zero. So, we have

$$\Pr_{r \in Z_p}[C(r) = 0] \leq (n - 1)/|Z_p| \leq n/n^2 = 1/n.$$  

So, in case $a \neq b$, Alice and Bob will decide that $a \neq b$ with probability at least $1 - 1/n$.

Note that by picking $p$ to be a larger number, e.g., at least $n^{100}$, we can make the error probability of this protocol smaller than $n^{-99}$. The communication complexity remains $O(\log n)$.

## 2 Randomized Complexity Classes

### 2.1 One-sided error

A language $L \in \text{RP}$ if there is a deterministic polytime TM $M(x, r)$ such that

1. for all $x \in L$, $\Pr_r[M(x, r) \text{ accepts}] \geq 1/2$, and
2. for all $x \notin L$, $\Pr_r[M(x, r) \text{ accepts}] = 0$.

Here, we assume some polynomial bound on $|r|$ in terms of $|x|$, i.e., for some constant $c$, we have $|r| \leq |x|^c$.

Now, to decide $L$, on input $x$, a randomized algorithm may first flip $|r|$ random coins to compute a random string $r$, and then simulate $M(x, r)$. This randomized algorithm will run in polytime, and will be correct for all $x \notin L$ with probability 1, and will be correct for all $x \in L$ with probability at least 1/2.
Note that if $L \in \mathsf{RP}$, then $L \in \mathsf{NP}$. So, we get that $\mathsf{RP} \subseteq \mathsf{NP}$. The class $\mathsf{RP}$ contains those languages that can be decided probabilistically with one-sided error: an algorithm may err on positive instances, but never on negative instances. Next, we define the class of languages decidable with two-sided error.

### 2.2 Two-sided error

A language $L \in \mathsf{BPP}$ if there is a polytime DTM $M(x,r)$ such that

1. for all $x \in L$, $\Pr_r[M(x,r) \text{ accepts}] \geq 3/4$, and
2. for all $x \notin L$, $\Pr_r[M(x,r) \text{ accepts}] \leq 1/4$.

Note that now we allow a “small” probability of error even on inputs not in the language.

### 2.3 Zero error

There is also a class $\mathsf{ZPP}$ of languages decidable by a zero-error probabilistic polytime algorithm. Such an algorithm runs in polytime, and either outputs a correct answer, or says “Don’t know”. The requirement is that the probability of the “Don’t know” answer be less that $1/3$.

More formally, a language $L \in \mathsf{ZPP}$ if there is a polytime DTM $M(x,r)$ such that for every $x$, and every $r$, $M(x,r)$ either outputs a correct answer $0$ (if $x \notin L$) or $1$ (if $x \in L$), or says $?$ (“I don’t know”). Moreover, for every $x$, we have $\Pr_r[M(x,r) =?] \leq 1/3$.

It is easy to show the following:

**Theorem 1.** $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$.

*Proof.* Take any language $L \in \mathsf{ZPP}$. Let $M(x,r)$ be a polytime DTM as in the definition of $\mathsf{ZPP}$. Consider the following algorithm: “On input $x$, run $M(x,r)$ for a random $r$. If $M$ accepts, then Accept. If $M$ rejects or outputs “?”, then Reject.” It is easy to see that the described algorithm is an $\mathsf{RP}$ algorithm for $L$. To get an $\mathsf{RP}$ algorithm for $\bar{L}$ we argue analogously.

For the other direction, given an $\mathsf{RP}$ algorithm $M_1$ for $L$, and an $\mathsf{RP}$ algorithm $M_2$ for $\bar{L}$, consider the following new algorithm: “On input $x$, pick independent random strings $r_1$ and $r_2$. Run $M_1(x,r_1)$ and $M_2(x,r_2)$. If $M_1$ accepts, then Accept. If $M_2$ accepts, then Reject. Otherwise, output “?”.” Clearly, the described algorithm either outputs a correct answer, or “?”. The latter happens with probability at most $(1/2)^2 < 1/3$, as the two events that $M_1$ and $M_2$ both reject are independent. 

Finally, one can also define randomized algorithm that run in expected polytime and never make an error (but sometimes may run for a very long time). Such an algorithm may be converted into a $\mathsf{ZPP}$-style algorithm. Before showing this, let us recall a useful fact about expectations of positive integer-valued random variables.
Lemma 2. Let $X \geq 0$ be any integer-valued random variable. Then

$$\text{Exp}[X] = \sum_{i=0}^{\infty} \text{Pr}[X > i].$$

Proof. Consider the sum:

$$\text{Pr}[X = 1] + \text{Pr}[X = 2] + \text{Pr}[X = 3] + \ldots$$

Looking at it “row by row”, we get $\sum_{i=0}^{\infty} \text{Pr}[X > i]$. Looking at it “column by column”, we get $\sum_{i=1}^{\infty} i \cdot \text{Pr}[X = i] = \text{Exp}[X]$. □

Theorem 3. $L \in \text{ZPP}$ iff there is an expected-polytime randomized algorithm for $L$ that never outputs a wrong answer.

Proof. First, suppose $L$ has a randomized algorithm that is always correct and runs in expected time $T = n^c$, for some $c > 0$. By Markov’s Inequality (see the lecture notes for the previous lecture), we know that the probability that our randomized algorithm runs for $4 \cdot T$ steps is at most $1/4$. So to get a ZPP algorithm for $L$, we run a given algorithm for $4 \cdot T$ steps, outputting its answer if we get an answer, and otherwise (if we time-out) output “?”. For the other direction, let $L \in \text{ZPP}$ for a polytime DTM $M(x, r)$. Consider the following algorithm: “On input $x$, until we get an answer other than ‘?’, keep running $M(x, r)$ with fresh randomness each time. Once we get the answer, output that answer and halt. (Otherwise, keep running $M$.)”

Obviously, the described algorithm never outputs a wrong answer. We just need to show it runs in expected polytime. Let $T$ be the random variable that is the running time of our described algorithm. Let $t = \text{poly}(n)$ be the runtime of $M(x, r)$, which is the running time of a single iteration of our described algorithm. Since each iteration uses fresh (independent) randomness, the probability our algorithm runs for more than $k$ iterations is at most $3^{-k}$. By Lemma 2, we get that the expected number of iterations of our algorithm is:

$$\sum_{i=0}^{\infty} \text{Pr}[\text{# iterations} > i] \leq \sum_{i=0}^{\infty} 3^{-i} = \frac{1}{1 - (1/3)} = \frac{3}{2}. $$

Hence, the expected runtime is $(3/2) \cdot t = \text{poly}(n)$, as required. □

3 Reducing the error probability

3.1 RP

Consider any $L \in \text{RP}$. Let $M(x, r)$ be the corresponding polytime DTM. We design a new DTM $M'(x; r_1, \ldots, r_l)$ such that $M'(x; r_1, \ldots, r_l)$ accepts iff there is some $1 \leq i \leq l$ such
Recall that a language \(L\) of that kind is known. We claim that the error probability of the new \(M'\) is at most \(2^{-l}\).

Indeed, if \(x \notin L\), then \(M(x, r_i)\) rejects for every \(i\), and so \(M'\) also rejects for all sequences \(r_1, \ldots, r_l\). On the other hand, if \(x \in L\), then \(\Pr_{r_1, \ldots, r_l}[M'(x; r_1, \ldots, r_l) \text{ rejects}]\) is equal to \(\prod_{i=1}^{l} \Pr_{r_i}[M(x, r_i) \text{ rejects}] \leq (1/2)^l\). So, in the case where \(x \in L\), the TM \(M'\) accept for a fraction \(1 - 2^{-l}\) of all random sequences \(r_1, \ldots, r_l\).

This ability to reduce error probability allows us to prove the following.

**Theorem 4.** \(\text{RP} \subseteq \text{BPP}\)

**Proof.** By the argument above, for any \(L \in \text{RP}\), there is a DTM \(M(x, r)\) such that, for every \(x \in L\), \(\Pr[M(x, r) \text{ accepts}] \geq 1 - 1/4 = 3/4\) — simply take \(l = 2\) in the above-mentioned error reduction procedure. On the other hand, for every \(x \notin L\), \(\Pr[M(x, r) \text{ accepts}] = 0 < 1/4\). So, \(L \in \text{BPP}\). \(\square\)

What about \(\text{NP}\) and \(\text{BPP}\)? Is one a subclass of the other? It is unknown! The general belief is that \(\text{BPP} = \text{P}\), and therefore, \(\text{BPP}\) is a subset of \(\text{NP}\). But, no unconditional result of that kind is known.

### 3.2 \(\text{BPP}\)

Recall that a language \(L\) is in \(\text{BPP}\) if there is a deterministic polytime TM \(M(x, r)\), such that for \(x \in L\), \(M(x, r)\) accepts at least \(3/4\) of \(r\)'s, and for \(x \notin L\), \(M(x, r)\) accepts at most \(1/4\) of all \(r\)'s. In each case, the probability that \(M\) makes a mistake (e.g., accepts \(x \notin L\)) is at most \(1/4\). The choice of this error probability to be \(1/4\) is rather arbitrary. As we show next, it is always possible to make the error probability exponentially small, by increasing the running time only slightly.

The idea is to choose \(l\) independent copies of a random string \(r\), i.e., chose independently uniformly at random \(r_1, r_2, \ldots, r_l\). Then simulate \(M(x, r_i)\), for each \(1 \leq i \leq l\), noting whether \(M\) accepts or rejects for each \(r_i\). Finally, our new algorithm will accept if and only if the majority (i.e., \(> l/2\)) of random strings \(r_i\)'s were accepted.

Intuitively, if \(x \in L\), then each \(r_i\) has the probability at least \(3/4\) of being accepted by \(M(x, r_i)\). So, on average, \(M\) will accept at least \(3/4\) of the strings in the list \(r_1, r_2, \ldots, r_l\). It seems very unlikely that the actual number of accepted \(r_i\)'s will deviate significantly from the expected number. (This is basically the Law of Large Numbers from Probability Theory.

**Theorem 5** (Chernoff bounds). Let \(X_1, \ldots, X_n\) be independent identically distributed random variables such that \(\Pr[X_i = 1] = p \text{ and } \Pr[X_i = 0] = 1 - p\), for some \(0 \leq p \leq 1\). Consider the new random variable \(X = \sum_{i=1}^{n} X_i\) with expectation \(\mu = pn\). Then, for any \(0 < \delta < 1\),

\[
\Pr[|X - \mu| > \delta \mu] < 2e^{-\delta^2 \mu/3}.
\]

Back to the analysis of our new algorithm that simulates \(M(x, r)\) on \(l\) independent copies of random string \(r_i\). Let’s define a random variable \(X_i\), for \(1 \leq i \leq l\), such that \(X_i = 1\) if \(M(x, r_i)\) produces a correct answer (i.e., accepts if \(x \in L\), and rejects if \(x \notin L\), and
$X_i = 0$ otherwise. Observe that, by definition of \( \text{BPP} \), we have $\Pr[X_i = 1] = p \geq 3/4$. Let's analyze the probability that our new algorithm makes a mistake, i.e., that the majority of the variables $X_1, \ldots, X_l$ are 0. Let $X = \sum_{i=1}^{l} X_i$, and let $\mu = pl \geq (3/4)l$ be the expectation of $X$. We have

$$\Pr[X < l/2] \leq \Pr[X < (2/3)\mu] = \Pr[|X - \mu| > \mu/3].$$

By the Chernoff theorem, the last probability is at most $2e^{-\mu/27} \leq 2e^{-l/36}$, which is exponentially small in $l$, the number of times we run the original algorithm $M$.

Thus, by taking $l = \text{poly}(n)$, where $n = |x|$, we can reduce the probability error of a new algorithm below an inverse exponential in $n$, while still running in polytime.

### 4 BPP and Small Circuits

The ability to reduce the error probability in \( \text{BPP} \) has a curious consequence that every language in \( \text{BPP} \) is computable by a family of polysize Boolean circuits.

**Theorem 6.** \( \text{BPP} \subset \text{P/poly} \)

**Remark 7.** Note that \( \text{P} \subset \text{P/poly} \) trivially, since a polytime deterministic TM can be simulated by a polysize Boolean circuit. However, it is not at all trivial to argue that every \( \text{BPP} \) algorithm can also be simulated by a small Boolean circuit. The problem seems to be: what do we do with random strings used by the \( \text{BPP} \) algorithm? Now, we'll prove the theorem, and thus explain what to do with all those random strings.

**Proof of Theorem 6.** Consider an arbitrary $L \in \text{BPP}$. Since we know how to reduce the error probability below any inverse exponential in the input size, we may assume that there is a deterministic polytime TM $M(x,r)$ such that, for every $x \in L$, $\Pr_r[M(x,r) = 1] > 1 - 2^{-2n}$, and for every $x \notin L$, $\Pr_r[M(x,r) = 1] < 2^{-2n}$, where $|x| = n$, and $|r|$ is some polynomial in $n$.

Now, $\Pr_r[\exists x \text{ s.t. } M(x,r) \text{ is wrong}] \leq 2^n \Pr_r[M(x,r) \text{ is wrong for a fixed } x] \leq 2^n 2^{-2n} = 2^{-n} \ll 1$. The first inequality is the so-called “union bound” saying that for any events $E_1, \ldots, E_n$, $\Pr[E_1 \lor E_2 \ldots E_n] \leq \sum_{i=1}^{n} \Pr[E_i]$. The second inequality uses the fact that $M$ has error probability at most $2^{-2n}$ for every input of size $n$.

Hence, there must exists at least one string $\hat{r}$ such that $M(x,\hat{r})$ is correct for every input $x$ of length $n$. We can use such a string $\hat{r}$ as an advice string, and then just simulate $M(x,\hat{r})$ on any given input $x$ of length $n$. Thus, $L \in \text{P/poly}$. \( \square \)