Introduction, Equality Testing, Monte Carlo[1]

1 What are Randomized Algorithms?

- Randomized algorithms are usual algorithms which have an extra resource available to them: they have access to a stream of *independent* fair random coins. More pertinently, we assume the algorithm has access to a subroutine $\mathsf{rand}()$, and each time the algorithm calls it, it obtains a bit which is 0 or a 1. The probability of seeing either answer is equal to $1/2$. Furthermore any two calls to $\mathsf{rand}()$ produces answers which are independent of each other, that is, the call of one $\mathsf{rand}()$ does not affect subsequent calls to $\mathsf{rand}()$. In Python 3, for example, one can `import random` and then use `random.randint(0,1)`.

- If one is encountering this concept for the first time, it may feel very counter-intuitive why access to "randomness" or "noise" can help in the design and analysis of algorithms. The main objective of this course is to show how randomness can be a powerful tool. We stress, and we will stress this many times in the course, this is true even in the *worst-case* analysis of algorithms. That is, we will still be operating under the assumption that our algorithm must behave well for *all* inputs to the problem. We will not be using randomness to analyze the working of the algorithm on a "random input".

- There is going to be one catch however. Our randomized algorithms will either make mistakes, or may run for ever. However, the *probability* of these "bad events" will be "extremely small", and in fact can be made as small as we want. I stress: this small can be as small as 1 in 2^{100}, and therefore one shouldn't be worried about this error. Nevertheless, it is not going to be 0.

  Remark: *Usually, at this point one can ask the question : "How does* $\mathsf{rand}()$ *get random bits? What exactly is random?" These are fascinating questions, but something we will probably not cover in this course. For us, we are going to accept the existence of random bits as an* ***axiom***.

2 Alice and Bob check for Equality

- Consider the following problem. Alice has a huge file which we are going to think of as a bit-string $\mathbf{x} \in \{0,1\}^n$. Bob also has a file $\mathbf{y} \in \{0,1\}^n$. They need to figure out if they have the same file, that is, is $\mathbf{x} = \mathbf{y}$? To do so, they must communicate with each other. What is the smallest amount of communication that will solve this problem?

- The trivial solution is : Alices send her file $\mathbf{x}$ to Bob who checks if $\mathbf{x} = \mathbf{y}$ and then tells the answer. This takes n bits of communication. Can one do better? It so happens that without randomness the answer[2] is "No!".

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 25th March, 2024*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

[2] More precisely, for any algorithm/protocol $\mathcal{A}$ that Alice and Bob decide on, if the algorithm communicates less than n bits, then there exists some input pair $(\mathbf{x}, \mathbf{y})$ where the algorithm answers incorrectly. This is not too hard to show, but is out of the scope of this course. But there is a whole other CS49/249 called "Communication Complexity" which answers such questions.

- Now suppose Alice and Bob can use randomness. Let us be precise in telling what this means. We are going to assume there exists a source of random bit $r_1, r_2, r_3, \ldots$ such that each r_i is 0 or 1 with probability $1/2$ and they are mutually independent. Furthermore, this source is *public* and both Alice and Bob can read it. Where can such a source be found? This relates to the remark above. For a working example, let us say these are the parities of the digits of π from a certain point. They clearly don't satisfy what we need, but they may not be a bad proxy.

- ***Algorithm.*** Here is what Alice does:

```
procedure ALICE-EQ($\mathbf{x}$):
    for $i = 1$ to $k$ do:▷ $k$ is a parameter we will set later.
        Get $n$ random bits $\mathbf{r}_i := (\mathbf{r}^{(i)}_1, \ldots, \mathbf{r}^{(i)}_n)$ from public random source.
        Compute $b_i := \mathbf{x} \cdot \mathbf{r}_i \bmod 2 = \sum_{j=1}^{n} \mathbf{x}_j \mathbf{r}^{(i)}_j \bmod 2$.
    Send $\mathbf{b} := (b_1, b_2, \ldots, b_k)$ to Bob. ▷ This is $k$ bits of communication.
```

Upon receiving the bits, here is what Bob does.

```
procedure BOB-EQ($\mathbf{b} \in \{0,1\}^k$, $\mathbf{y}$):
    for $i = 1$ to $k$ do:▷ $k$ is a parameter we will set later.
        Get the same $\mathbf{r}_i$ as Alice. ▷ Public Randomness
        Compute $c_i := \mathbf{y} \cdot \mathbf{r}_i \bmod 2$.
    If all $b_i = c_i$, answer EQUAL, otherwise answer UNEQUAL.
```

- ***Analysis.*** Let us start with a simple observation. If $\mathbf{x} = \mathbf{y}$, then Bob will ***always*** correctly answer EQUAL, as $\sum_{j=1}^{n} \mathbf{x}_j \mathbf{r}_j = \sum_{j=1}^{n} \mathbf{y}_j \mathbf{r}_j$ for *any* $\mathbf{r}_j$'s. Note that this needs Alice and Bob to have access to the *same* random bits.

  But what if $\mathbf{x} \neq \mathbf{y}$. Then does Bob always answer UNEQUAL? No. Suppose all the $\mathbf{r}_j$'s were 0. Then no matter what x and y are, the b_i's and c_i's will all be 0, and Bob will incorrectly answer UNEQUAL. However, the chances of that happening are pretty slim. That is, however, not the only way Bob can make a mistake. So, let us understand how Bob can err.

  Suppose $\mathbf{x} \neq \mathbf{y}$. Define $\mathbf{z} := \mathbf{x} - \mathbf{y}$, which is a *non-zero* vector. Note each coordinate of $\mathbf{z}$ is either -1 or 0 or 1. Here is the claim which bounds the error probability.

Claim 1. Let $\mathbf{z}$ be *any* non-zero n-dimensional $\{-1, 0, 1\}^n$ vector. Let $\mathbf{r}$ be a random n-dimensional vector with $r_i \in \{0, 1\}$ with probability $1/2$. Then,

$$\mathbf{Pr}[\mathbf{z} \cdot \mathbf{r} \bmod 2 = 0] \quad = \quad \frac{1}{2}$$

Proof. Since $\mathbf{z} \neq 0$, we know one of its coordinates is $\neq 0$. Without loss of generality let us assume it is $\mathbf{z}_1$. Then, let's write $\mathbf{z} \cdot \mathbf{r}$ as

$$\mathbf{z} \cdot \mathbf{r} = \sum_{i=1}^{n} \mathbf{z}_i \mathbf{r}_i = (\mathbf{z}_1 \mathbf{r}_1) + \sum_{i=2}^{n} \mathbf{z}_i \mathbf{r}_i$$

Let X be the random variable $\mathbf{z}_1\mathbf{r}_1$ and let $Y = \sum_{i=2}^{n} \mathbf{z}_i\mathbf{r}_i$. The *crucial* thing to note is that X and Y are independent random variables. We stress: even if $\mathbf{z}_1$ and $\mathbf{z}_2$ may be "dependent" in that they are set by the adversary, they are "fixed" and not random at all. The only randomness is in the $\mathbf{r}_i$'s, and $\mathbf{r}_1$ is independent of $\mathbf{r}_2, \ldots, \mathbf{r}_n$.

Let $Z = X + Y$ be the random variable $\mathbf{z} \cdot \mathbf{r}$. Below, we use $\equiv_2 0$ as a short hand of equal modulo 2.

$$\begin{aligned}
\mathbf{Pr}[Z \equiv_2 0] = \;& \mathbf{Pr}[Z \equiv_2 0 | Y \equiv_2 0] \cdot \mathbf{Pr}[Y \equiv_2 0] \;+\; \mathbf{Pr}[Z \equiv_2 0 \mid Y \equiv_2 1] \cdot \mathbf{Pr}[Y \equiv_2 1] \\
= \;& \mathbf{Pr}[X \equiv_2 0 | Y \equiv_2 0] \cdot \mathbf{Pr}[Y \equiv_2 0] \;+\; \mathbf{Pr}[X \equiv_2 1 \mid Y \equiv_2 1] \cdot \mathbf{Pr}[Y \equiv_2 1] \\
= \;& \underbrace{\mathbf{Pr}[X \equiv_2 0]}_{=1/2} \cdot \mathbf{Pr}[Y \equiv_2 0] \;+\; \underbrace{\mathbf{Pr}[X \equiv_2 1]}_{=1/2} \cdot (1 - \mathbf{Pr}[Y = 0]) \\
= \;& 1/2,
\end{aligned}$$

where the last equality follows from the observation that $\mathbf{Pr}[X \equiv_2 0] = \mathbf{Pr}[\mathbf{r}_1 = 0]$ because $\mathbf{z}_1 \neq 0$, which implies $\mathbf{z}_1 \in \{-1, 1\}$ implying $\mathbf{z}_1 \equiv_2 1$. The first equality is the total law of probabilities. To explain the second equality, note that when $Y \equiv_2 0$, we get $Z \equiv_2 0$ if and only if $X \equiv_2 0$, and when $Y \equiv_2 1$, we get $Z \equiv_2 0$ if and only if $X \equiv_2 1$. The third equality follows from independence of X and Y. □

The above claim shows that a single $\mathbf{r}$ catches inequality with probability $1/2$. To get low error, we **boost** by repeating. In particular, if we want an error of 2^{-300} (which is a really, really small number), then Alice needs to send only 300 bits.

Theorem 1. For any $\delta > 0$, Alice and Bob can solve the equality problem with error probability $\leq \delta$ communicating $k := \lceil \lg\left(\frac{1}{\delta}\right) \rceil$ bits.

Proof. We have already argued there is no error when $\mathbf{x} = \mathbf{y}$. When $\mathbf{x} \neq \mathbf{y}$, for every $1 \leq i \leq k$, the previous claim gives us $\mathbf{Pr}[b_i = c_i] = \frac{1}{2}$. Since all the random bits r_i's are independent, the events $\mathcal{E}_i := \{b_i = c_i\}$ are mutually independent. Therefore, the probability $\mathbf{Pr}[\bigwedge \mathcal{E}_i] = \frac{1}{2^k} \leq \delta$. This is precisely the probability Bob says EQUAL. □

Remark: *The presence of this public random bits might bother you. Perhaps you would prefer Alice to run an algorithm using* rand() *function on her computer, and then send whatever she wants to send to Bob. Bob can also run the* rand() *function on his computer, but they no longer can agree with Alice's. Can you think of an algorithm which works under this assumption? We will cover this later in the course.*

Ponder This: *How many random bits does the above algorithm use from the public source? Quite a lot, right. Can we do with **fewer** random bits? For example, would the following work: Alice and Bob decide on a **fixed** long string s (say the parities of the digits of π). We will assume $|s| \geq n^2$. Alice then chooses a random index $i \in \{1, 2, \ldots, n\}$, and then executes the above protocol with $\mathbf{r}$ set to $(s_i, s_{i+1}, \ldots, s_{i+n-1})$.*

- **Multiple Equalities: The Union Bound.** Consider a generalization of the problem. Alice now has T strings $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$ where each $\mathbf{x}^{(T)} \in \{0,1\}^n$. Bob still has that one string $\mathbf{y} \in \{0,1\}^n$. They want to know whether *there exists* and $1 \le t \le T$ such that $\mathbf{x}^{(t)} = \mathbf{y}$ or are they *all* different from $\mathbf{y}$. How much communication is needed if, say, they want to be correct with 99% probability?

  At first blush, one may assume that the answer is simply $O(T)$ since instead of sending $O(\log(1/\delta))$-bits, which is $O(1)$ for $\delta = 0.01$, Alice sends T many such copies. However, it is important to note that *every* such message has an error probability and error accumulates. This is what is captured by the so-called "union bound" which is a simple and yet powerful and ubiquitous concept. Let's spell it out for this case.

  Let $\mathcal{B}_t$ be the event that $\mathbf{x}^{(t)} \neq \mathbf{y}$ and yet the bits $\mathbf{b}^{(t)} = (b_1^{(t)}, \ldots, b_k^{(t)})$ that Alice sends, for $k = \lceil \lg(1/\delta) \rceil$ matches $\mathbf{c}^{(t)}$, the bits that Bob computes. That is, $\mathcal{B}_t$ is the event that the Equality test between $\mathbf{x}^{(t)}$ and $\mathbf{y}$ passes. For the above algorithm, the bad event occurs when *any* of the $\mathcal{B}_t$'s occurs. So,

$$\mathbf{Pr}[\text{algo errs}] \;=\; \mathbf{Pr}[\bigvee_{t=1}^{T} \mathcal{B}_t] \;\underbrace{\le}_{\text{Union Bound}}\; \sum_{i=1}^{T} \mathbf{Pr}[\mathcal{B}_t]$$

  Note that we have moved from an equality to an inequality; we are using the near trivial fact the probability an event $\mathcal{E}$ occurs or an event $\mathcal{E}'$ occurs is at most the sum of their individual probabilities. Oftentimes (but not always) we don't qualitatively lose much by making this over-estimation (and save a lot on sweat).

  Since we know that $\mathbf{Pr}[\mathcal{B}_t] \approx \delta$, we see that all we can say of the error probability of our algorithm is that it is $\le T\delta$. And so, if we want this to be ≤ 0.01, we need to make $\delta = 0.01/T$, which in turn means that Alice would need to send $O(T \log T)$ bits.

  Ponder This: *Can you actually solve this in $O(T)$ bits?*

3 Monte Carlo Algorithms

- More generally, a randomized algorithm $\mathcal{A}$ on an input I could run in some *fixed, deterministic* time $T_{\mathcal{A}}(I)$, but return a solution $\mathcal{A}(I)$ which *could* be ***wrong***. However, the probability that the algorithm is wrong (the probability taken over the randomness generated by the algorithm's various calls to `rand`()) is "small". Formally, for our lectures let's assert that

$$\mathbf{Pr}[\mathcal{A}(I) \;\text{ is wrong}] \le \frac{1}{3}$$

  These are called *Monte Carlo* randomized algorithms.

- Let us be a bit more precise about the computational problems at hand. Most computation problems are either decision problem (with Yes or No answers), or optimization problems (which tries to maximize or minimize some function). In both cases, we could *ramp down* the error probability pretty rapidly by *repeating*. Let us elaborate for decision problems. Suppose the problem was a decision problem, and suppose $\mathcal{A}$ took input $\mathcal{I}$ and returned a solution YES or NO. For decision problems, the Monte-Carlo algorithms are of two types: one with one-sided error and the other with two sided-error.

- *One-sided error.* The algorithm $\mathcal{A}$ is a Monte-Carlo algorithm with one-sided error, if for any instance $\mathcal{I}$ whose answer is YES, the algorithm answers YES with probability 1. When the instance $\mathcal{I}$ has answer NO, the algorithm may answer YES with probability ≤ 0.1. Given such an algorithm $\mathcal{A}$ and an instance $\mathcal{I}$ imagine what happens if we repeat $\mathcal{A}$ on $\mathcal{I}$ for k times each time *using fresh random coins*. If $\mathcal{I}$ was an YES instance, the algorithm will return YES all the time. If $\mathcal{I}$ was a NO instance, however, the algorithm could make mistakes. But the probability that the algorithm makes a mistake all k times is $\leq \frac{1}{3^k}$. This suggests the following ***boosting algorithm***: run $\mathcal{A}$ on the instance k times and return YES if all k runs return YES, otherwise, return NO.

  If $\mathcal{I}$ is a YES instance, the boosting algorithm says YES with probability 1. If $\mathcal{I}$ is a NO instance, the boosting algorithms says YES with probability $\leq \frac{1}{3^k}$. Note that if $k = 30$, this number is $\leq 10^{-14}$.

  Remark: *How would you ramp error down for optimization problems? Suppose that the algorithm always returns a* feasible *solution with probability* 1.

- *Two-sided error.* The algorithm $\mathcal{A}$ is a Monte-Carlo algorithm with two-sided error, if it could make an error on both YES and NO instances. However, in either case the chance of error is $\leq \frac{1}{3}$. First notice that the above method of boosting won't work. Rather, what works is the following: repeat for $2k + 1$ times the run of $\mathcal{A}$ on $\mathcal{I}$, and return the *majority* answer. In a later lecture, we will see how the error probability of this repeated algorithm also exponentially decays with k.

Learning Tidbits:

- Algorithm Design: *"dot-product with a random vector" as a aggregation/summarizing rule. For another use of this idea, see the "checking matrix-multiplication" problem in the problem set.*
- Analysis: *Use of conditional probability; Boosting; Union Bound. The use of union bound is going to be prevalent throughout the course.*