**CSE525: Randomized Algorithms and Probabilistic Analysis** April 2, 2013

# Lecture 1

*Lecturer: Anna Karlin* *Scribe: Sonya Alexandrova and Eric Lei*

## 1 Introduction

The main theme of this class is randomized algorithms. We start by comparing these to the deterministic algorithms to which we are so accustomed. In the deterministic model of computation (Turing machines and RAM), an algorithm has fixed behavior on every fixed input. In contrast, in the randomized model of computation, algorithms take additional input consisting of a stream of random bits. Their behavior and performance can therefore vary on a fixed input. To illustrate the difference, consider quicksort where the input is a random permutation but the pivot is always the first element, and compare it to quicksort with a random pivot. In the first case, even though the input is a random variable, quicksort will always execute the same behavior. Consequently, the algorithm is deterministic. In the second case, however, multiple executions could produce different behavior on the same input, so the algorithm is randomized, and performance is a random variable.

Naturally, we question whether studying randomized algorithms is worth the effort. The answer is a resounding yes, since they are often the simplest or fastest known algorithms, as we shall soon see. Also, they are fun!

This lecture will showcase several classical examples of randomized algorithms. These notes are organized as follows: Section 1—Introduction; Section 2—Matrix-Product Verification; Section 3—Fingerprinting; Section 4—Max Cut.

## 2 Matrix-Product Verification

In this section, we will cover one of the first published uses of randomization in algorithms: Freivald's algorithm (1977). Simple and elegant, the algorithm verifies the product of square matrices, given a candidate for the solution. More precisely, given $(n \times n)$ matrices $A$, $B$, and $C$ over a field $F$, we must decide whether the equation $AB = C$ holds. The obvious deterministic method to verify a matrix product is to simply compute $AB$ and compare it to $C$. The runtime is $O(n^{2.3727})$, using the Coppersmith-Winograd algorithm for matrix multiplication. Yet we will prove that Freivald's algorithm (Algorithm 1) does better, with a high probability of correctness.

A field $F$ is a set over which addition and multiplication are defined with all the properties to which we are accustomed, including commutativity, associativity, inverses, and identity elements. In the discussion that follows, we take $F = \text{GF}(2)$, the Galois field of two elements. Informally, GF(2) is akin to the set of integers modulo 2. In computer science, GF(2) is commonly used because it corresponds to bitwise arithmetic: addition is the XOR operation, and multiplication is the AND operation.

**Algorithm 1** Freivald's algorithm

1: **input:** $(n \times n)$ matrices $A$, $B$, and $C$ over GF(2)
2: $r \leftarrow (r_1, r_2, \ldots, r_n) \in \{0,1\}^n$ where the $\{r_i\}$ are independent, and each is equally likely to be 0 or 1
3: $z \leftarrow A(Br)$ // *computing $Br$ first avoids multiplying two matrices*
4: **if** $Cr = z$ **then**
5: **return** "yes, $AB = C$"
6: **else**
7: **return** "no, $AB \neq C$"
8: **end if**

To summarize the algorithm, we draw a random vector of $n$ *i.i.d.* Bernoulli random variables with parameter $p = 1/2$. Next we compute the vector $ABr$, by first computing $Br$ and multiplying the result by $A$. Then we output the result of checking $ABr = Cr$, a necessary but not sufficient condition of $AB = C$.

Now we analyze the correctness and runtime of this algorithm.

**Claim 1.** *Let $E$ denote the event of outputting an incorrect answer; i.e., $E = \{ABr = Cr \wedge AB \neq C\}$. Then $\Pr(E) \leq 1/2$.*

**Proof** Let $D = AB - C$. Then $D = 0$ if and only if $AB = C$. Suppose $D \neq 0$. Then there exists a nonzero entry $(i, j)$ of $D$, say $d_{ij}$. Let $r_{-j} = (r_1, \ldots, r_{j-1}, r_{j+1}, \ldots, r_n)$. We have

$$\begin{aligned}
\Pr(E) &= \Pr(Dr = 0 \wedge D \neq 0) && (1)\\
&\leq \Pr(Dr = 0) && (2)\\
&\leq \Pr(\sum_k d_{ik} r_k = 0) && (3)\\
&= \Pr(d_{ij} r_j = -\sum_{k \neq j} d_{ik} r_k) && (4)\\
&= \Pr(r_j = -\frac{1}{d_{ij}} \sum_{k \neq j} d_{ik} r_k) && (5)\\
&= \sum_{x \in \{0,1\}^{n-1}} \Pr(r_j = -\frac{1}{d_{ij}} \sum_{k \neq j} d_{ik} r_k | r_{-j} = x) \Pr(r_{-j} = x) && (6)\\
&\leq \sum_{x \in \{0,1\}^{n-1}} \frac{1}{2} \Pr(r_{-j} = x) && (7)\\
&= \frac{1}{2} && (8)
\end{aligned}$$

Line (3) states that $Dr = 0$ is no more likely than the event that the $i$th element of $Dr$ is 0, which is true because the former implies the latter. In Line (6), we condition on the value of $r_{-j}$ using the law of total probability. The inequality in Line (7) is true because the $\{r_i\}$ are independent and so $\{r_j = -\frac{1}{d_{ij}} \sum_{k \neq j} d_{ik} r_k | r_{-j} = x\}$ is the event that $r_j$ equals some particular value independent of the distribution of $r_j$. This probability equals 1/2 if the value is 0 or 1 and 0 otherwise. ■

**Remark** This proof illustrates the simple yet powerful *principle of deferred decisions*. When an analysis includes multiple random variables, we can think of instantiating some of them first and deferring the remainder. Formally, we use the law of total probability to condition on the values of the variables that we wish to set first; then we continue the analysis with just the remaining random variables. This principle can be seen in Lines (4) and (5), where we treat $r_j$ as the sole random variable in order to obtain the upper bound of 1/2.

**Remark** In the general setting of $F = \mathrm{GF}(d)$, we can improve this bound to $1/d$. We simply choose $r$ from $\{0, 1, \ldots, d-1\}^n$—the obvious adaptation. Then the 1/2 in Line (5) is replaced by $1/d$.

**Remark** The written lecture notes contain an error because they condition on $r_{-i}$ rather than $r_{-j}$, but $r_{-i}$ is insignificant.

**Claim 2.** *Algorithm 1 runs in $O(n^2)$ time.*

**Proof** It takes linear time to generate $r$. The matrix operations we perform are, in order, $Br$, $A(Br)$, and $Cr$. These are all multiplication of an $(n \times n)$ matrix by a vector of size $n$, which takes quadratic time. It takes linear time to check $Cr = z$. ■

There is a simple but highly effective way to improve the chance of correctness. We run the algorithm $k$ times and output a final "yes" if every iteration resulted in a "yes." Then $\Pr(error) \leq 1/2^k$ because the random bits in each trial are independent and so the outcomes are independent. However, the algorithm is now $O(kn^2)$. Thus, we gain an exponential improvement in correctness at a linear cost in runtime. This technique can be applied generally, and we will often use it in this class.

In sum, we gained a speed-up for verifying matrix products by using matrix-by-vector multiplication rather than matrix multiplication. The probability of correctness is the probability that it correct to check a necessary condition.

# 3 Fingerprinting

This section describes an application of randomized algorithms to a problem in information theory: the fingerprinting problem. This problem involves two agents $A$ and $B$ who own large databases and who are separated by a long distance. They would like to check whether their databases are equal. However, communication is costly, so they want to use as little communication as possible. Formally, given $n$-bit strings $a$ and $b$ belonging to $A$ and $B$ respectively, we must check whether $a = b$ with the minimum number of bits of communication between $A$ and $B$.

It is easy to see that an optimal, always correct deterministic algorithm is for $A$ to send $a$ to $B$, allowing $B$ to directly check $a = b$. This takes $n$ bits of communication. With randomization, however, we need significantly fewer bits (Algorithm 2).

**Algorithm 2** Fingerprinting algorithm

1: **input:** agents $A$ and $B$, their $n$-bit strings $a$ and $b$
2: $A$ chooses a prime $p \in [2 \dots x]$ uniformly at random, where $x = cn \ln n$ // *$c$ explained in Claim 3*
3: $A$ sends $(p, a \bmod p)$ to $B$
4: $B$ computes $b \bmod p$
5: $B$ returns the result of checking $a \bmod p \equiv b \bmod p$

To summarize the algorithm, a prime $p$ is selected randomly; $a$ and $b$ are compared modulo $p$. The only communication is $A$ sending $p$ and $a \bmod p$ to $B$.

**Claim 3.** *For any $c > 0$, we can configure Algorithm 2 to be incorrect with probability at most $1/c + o(1)$, where $o(1)$ is with respect to $n$. This $c$ is the same as that in Line 2 in Algorithm 2.*

**Proof** Algorithm 2 gives an incorrect answer in the case $a \neq b$ but $a \bmod p \equiv b \bmod p$. We bound the probability of this event.

$$\Pr(a \bmod p \equiv b \bmod p) = \Pr(p \text{ divides } a - b) \tag{9}$$

$$= \frac{\text{number of distinct primes that divide } a - b}{\text{number of primes in } [2 \dots x]} \tag{10}$$

$$\leq \frac{n}{x/\ln x} = \frac{n \ln x}{x} \tag{11}$$

$$= \frac{n \ln x}{cn \ln n} = \frac{1}{c} \frac{\ln x}{\ln n} \tag{12}$$

$$= \frac{1}{c} \frac{\ln(cn \ln n)}{\ln n} = \frac{1}{c} \frac{\ln n + \ln(c \ln n)}{\ln n} = \frac{1}{c}(1 + \frac{\ln(c \ln n)}{\ln n}) \tag{13}$$

$$= \frac{1}{c} + \frac{\ln(c \ln n)}{c \ln n} = \frac{1}{c} + o(1) \tag{14}$$

Line (10) states that the probability of picking a random prime that divides $a-b$ is equal to the ratio between the number of distinct primes that divide $a-b$ and the number of primes we are choosing from when we pick our random prime $p$. This holds because we pick $p \in [2 \dots x]$ uniformly at random.

Line (11) is obtained by approximating the numerator and denominator in Line (10). Since $a$ and $b$ are $n$-bit strings, $a-b$ is as well, so $a-b \leq 2^n$. Since prime numbers are $\geq 2$, the number of distinct primes that divide $a-b$ (the numerator) is no greater than $n$ because if we multiply together more than $n$ numbers that are at least 2, then we get a number greater than $2^n$. To approximate the denominator, we invoke the prime number theorem, which states that the number of primes less than or equal to $x$ is roughly $\frac{x}{\ln x}$.

In Lines (12) and (13), we use the fact that $x = cn \ln n$. ■

**Remark** The written notes contain an error in the statement of the prime number theorem by giving the formula as $x/\log_2 x$; the natural logarithm should be used instead. Fixing this error eliminates the need for the constant 1.26.

According to the above claim, when $c$ is large, then the algorithm has a high chance of being correct. The rationale behind this is that $x$ is proportional to $c$, so $c$ large implies a larger range $[2 \dots x]$ from which to pick $p$. Thus, larger $c$ means that it is less likely that $p$ will be a factor of $a-b$.

**Claim 4.** *Algorithm 2 uses $O(\log n)$ bits of communication.*

**Proof** The numbers transmitted are $p$ and $a \bmod p$, which are each at most $x$. Since $x$ is $\log_2 x$ bits, the number of bits transmitted is at most $2\log_2 x = 2\log_2(cn \ln n) = O(\log(n \log n)) = O(\log(n^2)) = O(\log n)$. ■

In a nutshell, the randomized fingerprinting algorithm gives an exponential decrease in the amount of communication at an arbitrarily small cost in correctness, by using a more efficient representation of the data—modular arithmetic—in exchange for a small loss of information.

# 4 Max Cut

This section demonstrates how randomization can lead to amazingly elegant proofs of nonrandom facts. In graph theory, a cut $C = (S, \bar{S})$ in a graph $G = (V, E)$ is a partition of $V$ into sets $S$ and $\bar{S}$. An edge $(u, v) \in E$ crosses $C$ if $u$ and $v$ belong to different sets. Let $\Gamma_C \subseteq E$ denote the set of edges that cross $C$. By giving a randomized algorithm (Algorithm 3), we can prove the following theorem about cuts in a surprisingly simple manner. It can be proved without any notion of randomness as well, but to do so is much harder by comparison.

**Algorithm 3** Random cut algorithm

```
1: input: a graph G = (V, E)
2: S ← ∅, S̄ ← ∅
3: for v ∈ V do
4:     flip a fair coin with outcome O
5:     if O = Heads then
6:         add v to S
7:     else
8:         add v to S̄
9:     end if
10: end for
11: return (S, S̄)
```

To summarize the algorithm, the vertexes are placed in $S$ or $\bar{S}$ independently and uniformly at random.

**Theorem 5.** *In any graph $G = (V, E)$, there exists a cut $C$ such that $|\Gamma_C| \geq |E|/2$.*

**Proof** The key idea is that in a random cut, the expected number of crossing edges equals $|E|/2$, so there must be some cut with at least $|E|/2$ crossing edges.

First we execute Algorithm 3 to obtain a cut $C = (S, \bar{S})$. For any edge $e \in E$, $\Pr(e \in \Gamma_C) = 1/2$ because there are four equally likely ways for the endpoints of $e$ to be classified, and the endpoints are in different sets in exactly two of these ways. To be explicit, if $e = (u, v)$, then we could have $u, v \in S$; $u \in S$ and $v \in \bar{S}$; $u \in \bar{S}$ and $v \in S$; or $u, v \in \bar{S}$. Since the classifications of $u$ and $v$ are independent, each of these outcomes is equally likely.

For any edge $e$, let $X_e = I\{e \in \Gamma_C\}$, where $I\{\cdot\}$ is the indicator function. Let $X = \sum_{e \in E} X_e$ denote the number of crossing edges. Then $E[X] = \sum_{e \in E} E[X_e] = \sum_{e \in E} \Pr(e \in \Gamma_C) = \sum_{e \in E} 1/2 = |E|/2$.

Therefore, there exists a possible outcome of the algorithm that returns a cut $C$ such that $|\Gamma_C| \geq |E|/2$. This is because the expectation is taken with respect to the possible outcomes. If every outcome led to a cut with fewer than $|E|/2$ crossing edges, then the average number of crossing edges would also be fewer than $|E|/2$. ■

**Remark** This proof is a typical example of a certain probabilistic technique. In general, to prove the existence of an object $O$ with property $P$, we show that $\Pr(\exists O \text{ with } P) > 0$. Here we did so by using the fact that not every item can be below (or above) the average. This implied that Algorithm 3 has a positive probability of returning a cut with the desired property.

In sum, we analyzed the behavior of a simple randomized algorithm to construct a proof a non-obvious fact about graph cuts. The main takeaway is the sheer elegance of the analysis.