

## Chapter 10: Randomization

Anup Rao

April 27, 2022

### Probability Spaces

A *probability space* is a set  $\Omega$  such that every element  $a \in \Omega$  is assigned a number  $0 \leq \Pr[a] \leq 1$  (called the probability of  $a$ ), and  $\sum_{a \in \Omega} \Pr[a] = 1$ .

An *event* in this space is a subset  $E \subseteq \Omega$ . The probability of the event is  $\sum_{a \in E} \Pr[a]$ . For example, imagine we toss a fair coin  $n$  times. Then the probability space consists of the  $2^n$  possible outcomes of the coin tosses. If  $E$  is the event that the first  $k$  coin tosses are heads, this event has probability exactly  $2^{-k}$ . Given two events  $E, E'$ , we write  $\Pr[E|E']$  to denote  $\Pr[E \cap E'] / \Pr[E']$ . This is the probability that  $E$  happens given that  $E'$  happens. We say that  $E, E'$  are independent if  $\Pr[E \cap E'] = \Pr[E] \cdot \Pr[E']$ . In other words,  $E, E'$  are independent if  $\Pr[E|E'] = \Pr[E]$ .

A *real valued random variable* is a function  $X : \Omega \rightarrow \mathbb{R}$ . The number of heads in the coin tosses is a random variable. The expected value of a random variable  $X$  is defined as  $\mathbb{E}[X] = \sum_{a \in \Omega} \Pr[a] \cdot X(a)$ . The following lemma is a very useful fact about random variables.

**Lemma 1** (Linearity of expectation). *If  $X, Y$  are real random variables, then  $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$ .*

#### Proof

$$\begin{aligned} \mathbb{E}[X + Y] &= \sum_{a \in \Omega} \Pr[a] \cdot (X(a) + Y(a)) \\ &= \sum_{a \in \Omega} \Pr[a] \cdot Y(a) + \sum_{a \in \Omega} \Pr[a] \cdot X(a) \\ &= \mathbb{E}[Y] + \mathbb{E}[X]. \end{aligned}$$

■

For example, let us calculate the expected number of runs of seeing 7 contiguous heads or tails in a 200 coin tosses. Let  $X_i$  be 1 if there are 7 heads or tails that start at the  $i$ 'th position, and 0 otherwise. If  $1 \leq i \leq 194$ , then  $\mathbb{E}[X_i] = \Pr[X_i = 1] = 2 \cdot 2^{-7} = 1/64$ . If  $i \geq 196$ , then  $X_i = 0$ . On the other hand, the total number of such runs is  $\sum_{i=1}^{194} X_i$ . So by linearity of expectation, the expected number of such runs is  $194/64 \approx 3.031$ .

In class, we discussed the waiting time to see the first heads. Suppose you keep tossing a fair coin until you see heads. Let  $T$  be the

Here is an expectation basic magic trick: Tell your audience to generate two sequences of coin tosses—one generated using 200 flips of a coin, and the second generated by hand. You leave the room, and they write both sequences on a black board. Then you come back into the room and immediately point out the sequence that was generated by hand. The trick: a random sequence is very likely to have a run of 7 heads or tails, while people tend to not insert such a long run into a sequence that they think looks random.

number of tosses you make. What is the expected value of  $T$ ? The key observation is that if the first toss is a heads, you stop with  $T = 1$ . Otherwise, the rest of the experiment is exactly the same as the original random experiment. So, we get:

$$\begin{aligned}\mathbb{E}[T] &= (1/2) \cdot 1 + (1/2) \cdot (1 + \mathbb{E}[T]) \\ \Rightarrow \mathbb{E}[T] \cdot (1 - 1/2) &= 1 \\ \Rightarrow \mathbb{E}[T] &= 2.\end{aligned}$$

### *Randomized Algorithms*

We shall give a few examples of problems where randomness helps to give very effective solutions.

#### *Matrix Product Checking*

Suppose we are given three  $n \times n$  matrices  $A, B, C$ , and want to check whether  $A \cdot B = C$ . One way to do this is to just multiply the matrices, which will take much more than  $n^2$  time. Here we give a randomized algorithm that takes only  $O(n^2)$  time.

**Input:** 3  $n \times n$ -matrices  $A, B, C$   
**Result:** Whether or not  $A \cdot B = C$ .  
 Sample an  $n$  coordinate column vector  $r \in \{0, 1\}^{0,1}$  uniformly at random ;  
**if**  $A(Br) = C(r)$  **then**  
 | Output "Equal";  
**else**  
 | Output "Not equal";  
**end**

**Algorithm 1:** Algorithm for Multiplication Checking

The algorithm only takes  $O(n^2)$  time. For the analysis, observe that if  $AB = C$ , then the algorithm outputs "Equal" with probability 1. If  $AB \neq C$ , the algorithm outputs "Equal" only when  $ABr = Cr \Rightarrow (AB - C)r = 0$ . We shall show that this happens with probability at most  $1/2$ .

Let  $D = AB - C$ . Then  $D \neq 0$ , so let  $d_{ij}$  be a non-zero entry of  $D$ . Then we have that the  $i$ 'th coordinate  $(Dr)_i = \sum_k d_{ik} \cdot r_k$ . This

coordinate is 0 exactly when  $r_j = (1/d_{ij}) \sum_{k \neq j} d_{ik} r_k$ . Finally, observe

$$\begin{aligned} & \Pr \left[ r_j = (1/d_{ij}) \sum_{k \neq j} d_{ik} r_k \right] \\ &= \sum_a \Pr \left[ a = (1/d_{ij}) \sum_{k \neq j} d_{ik} r_k \right] \cdot \Pr \left[ r_j = a \mid a = (1/d_{ij}) \sum_{k \neq j} d_{ik} r_k \right] \\ &\leq 1/2 \sum_a \Pr \left[ a = (1/d_{ij}) \sum_{k \neq j} d_{ik} r_k \right] \\ &= 1/2. \end{aligned}$$

**Exercise:** Modify the above algorithm so that the probability the algorithm outputs “Equal” when  $AB \neq C$  is at most  $1/4$ .

### 2-SAT

A two SAT formula is a CNF formula where each clause has exactly 2-variables. Here we give a randomized algorithm that can find a satisfying assignment to such a formula, if one exists.

**Input:** A two sat formula  $\phi$   
**Result:** A satisfying assignment for  $\phi$  if one exists  
 Set  $a = 0$  to be the  $n$ -bit all 0 string;  
**for**  $i = 1, 2, \dots, 100n^2$  **do**  
     **if**  $\phi(a) = 1$  **then**  
         Output  $a$ ;  
     **end**  
     Let  $a_i, a_j$  be the variables of an arbitrary unsatisfied clause.  
     Pick one of them at random and flip its value ;  
**end**  
 Output “Formula is not satisfiable”;

**Algorithm 2:** Algorithm for 2 SAT

If  $\phi$  is not satisfiable, then clearly the algorithm has a correct output. Now suppose  $\phi$  is satisfiable and  $b$  is a satisfying assignment, so  $\phi(b) = 1$ . We claim that the algorithm will find  $b$  (or some other satisfying assignment) within  $100n^2$  steps with high probability. To understand the algorithm, let us keep track of the number of coordinates that  $a, b$  disagree in during the run of the algorithm. Observe that during each run of the for loop, the algorithm picks a clause that is unsatisfied under  $a$ . Since  $b$  satisfies this clause,  $a, b$  must disagree in one of the two variables of this clause. Thus the algorithm reduces the distance from  $a$  to  $b$  with probability  $1/2$ .

Thus we can think of the algorithm as doing a random walk on the

line. There are  $n + 1$  points on the line, and at each step, if the algorithm is at position  $i$  it moves to position  $i + 1$  with probability  $1/2$  and to position  $i - 1$  with probability at least  $1/2$ . We are interested in the expected time before the algorithm hits position 0. Let

$$t_i = \mathbb{E} [\# \text{ steps before hitting position 0 from position } i].$$

Then we have the following equations:

$$\begin{aligned} t_0 &= 0, \\ t_i &= (1/2)t_{i+1} + (1/2)t_{i-1} + 1 \quad i \neq 0, n \\ \Rightarrow t_i - t_{i-1} &= t_{i+1} - t_i + 2 \\ t_n &= 1 + t_{n-1}. \end{aligned}$$

Thus we can compute:

$$\begin{aligned} t_n &= (t_n - t_{n-1}) + (t_{n-1} - t_{n-2}) + \dots + (t_1 - t_0) \\ &= 1 + 3 + \dots \\ &= \sum_{j=1}^n (2j - 1) = 2 \left( \sum_{j=1}^n j \right) - n = n(n + 1) - n = n^2. \end{aligned}$$

Thus the expected time for the algorithm to find a satisfying assignment is  $n^2$ .

**Lemma 2.**

$$\Pr[\text{algorithm does not find satisfying assignment in } 100n^2 \text{ steps}] < 1/100.$$

**Proof** We have that

$$\begin{aligned} n^2 &\geq \mathbb{E} [\# \text{ steps to find assignment}] \\ &= \sum_{s=0}^{\infty} s \cdot \Pr[s \text{ steps to find assignment}] \\ &\geq \Pr[\text{at least } 100n^2 \text{ steps are taken}] \cdot 100n^2. \end{aligned}$$

Therefore,

$$\Pr[\text{more than } 100n^2 \text{ steps are taken}] < 1/100.$$

■

*Max Cut*

Given a graph  $G = (V, E)$ , a subset  $S \subset V$  is called a cut of the graph. The size of the cut is the number of edges that cross from  $S$  to  $V - S$ .

It is known to be NP-hard to compute the MAX-cut of a graph. Here we give a simple randomized algorithm that will compute a cut that is half as big as the biggest cut in expectation.

The algorithm is just to pick the subset  $S$  at random, by including every vertex in  $S$  with probability half. For each edge  $e$ , let  $X_e$  be the random variable that is 1 if  $e$  goes from  $S$  to  $V - S$ , and 0 otherwise. Then we see that the size of the cut is exactly  $\sum_{e \in E} X_e$ . We can compute  $\mathbb{E}[X_e] = 1/2$ , and so by linearity of expectation,

$$\mathbb{E} \left[ \sum_{e \in E} X_e \right] = \sum_{e \in E} \mathbb{E}[X_e] = |E|/2.$$

### Fingerprinting

Suppose Alice has an  $n$ -bit string  $x$  and Bob has an  $n$ -bit string  $y$ , and they want to check that they are equal. Naively this takes  $n$  bits of communication between them. We can do much better using randomization.

Alice samples a random prime number  $p$  from the set of primes that are less than  $cn \ln n$ , for some constant  $c$  that we shall pick later. She then sends  $p$  and  $x \bmod p$  to Bob. Bob checks that  $x \bmod p$  is equal to  $y \bmod p$ . Thus they only need to communicate  $O(\log n)$  bits in this process.

If  $x = y$ , this will always produce the right outcome. We shall argue that if  $x \neq y$ , the probability that they make a mistake is going to be very small. To do this, we need a theorem:

**Theorem 3** (Prime number theorem). *Let  $\pi(a)$  denote the number of primes that are at most  $a$ . Then  $\lim_{a \rightarrow \infty} \frac{\pi(a)}{a/\ln a} = 1$ .*

When  $x \neq y$ , the above process fails only when  $p$  divides  $x - y$ . Since  $|x - y| \leq 2^n$ ,  $x - y$  can have at most  $n$  prime factors. On the other hand, by the prime number theorem, the number of primes of size up to  $cn \ln n$  is at least  $cn \ln n / (\ln(cn \ln n)) = \Omega(cn)$ . Thus the probability that the prime Alice picks divides  $x - y$  is at most  $O(1/c)$ .