6.080/6.089 GITCS 20 March 2008

Lecture 13

Lecturer: Scott Aaronson *Scribe: Emilie Kim*

1 Slippery Nature of Probabilities

Why are probabilities so counter-intuitive? Let's look at some examples to hone our intuition for probabilities.

1.1 Two children

Let's assume that boys and girls are equally likely to be born. A family has two children. One of them is a boy. What is the chance that the other one is a boy?

Older child:	B	G	B
Younger child:	G	B	B

The probability that the other child is a boy is 1/3, not 1/2.

The assumption in this question was that "One of them is a boy" meant that at least one of the children was a boy. If the question had instead been phrased such that a specific child was a boy, for example, "The older child is a boy. What is the probability that the younger one is a boy?", then the probability would have been 1/2.

1.2 Three unfortunate prisoners

There are three prisoners. One is randomly selected to be executed the next morning, but the identity of the unlucky prisoner is kept a secret from the prisoners themselves. One of the prisoners begs a guard, "At least tell me which of the other two prisoners *won't* be executed. I know at least one of them won't anyway." But the guard says, "From your perspective, that would increase your own chance of being executed from 1/3 to 1/2."

Is the guard correct? No, he isn't. The chance of being executed is still 1/3. (Why?)

1.3 Monty Hall

Suppose you are on a game show. There are three doors, of which one has a car behind it, and the other two have goats. The host asks you to pick a door; say you pick Door #1. The host then opens another door, say Door #3, and reveals a goat. The host then asks you if you want to stay with Door #1 or switch to Door #2. What should you do?

In probability theory, you always have to be careful about unstated assumptions. Most people say it doesn't matter if you switch or not, since Doors #1 and #2 are equally likely to have the car behind them. But at least in the usual interpretation of the problem, this answer is not correct.

The crucial question is whether the *host* knows where the car is, and whether (using that knowledge) he always selects a door with a goat. It seems reasonable to assume that this is the host's behavior. And in that case, switching doors increases the probability of finding the car from 1/3 to 2/3.

To illustrate, suppose the car is behind Door #1 and there are goats behind Doors #2 and #3.

Scenario 1: You choose Door #1. The host reveals a goat behind either Door #2 or Door #3. You switch to the other door and get a goat.

Scenario 2: You choose Door #2. The host reveals a goat behind Door #3. You switch to Door #1 and get the car.

Scenario 3: You choose Door #3. The host reveals a goat behind Door #2. You switch to Door #1 and get the car.

Here's another way of thinking about this problem. Suppose there are 100 doors, and the host reveals goats behind 98 of the doors. Should you switch? Of course! The host basically just told you where the car is.

1.4 Two envelopes

Suppose you have the choice between two envelopes containing money. One envelope has twice as much money as the other envelope. You pick one envelope, and then you're asked if you want to switch envelopes. The obvious answer is that it shouldn't make any difference. However, let x be the number of dollars in the envelope you originally picked. Then the expected number of dollars in the other envelope would seem to be $\frac{1}{2}(2x + \frac{x}{2}) = \frac{5x}{4}$, which is greater than x! So, can you really increase your expected winnings by switching? If so, then what if you keep switching, and switching, and switching—can you get an unlimited amount of money?

As a sanity check, suppose one envelope had a random number of dollars between 1 and 100, and the other envelope had twice that amount. Then if you believed that *your* envelope had more than 100 dollars, it's clear that you wouldn't switch. This simple observation contains the key to resolving the paradox.

See, the argument for switching contained a hidden assumption: that for every positive integer x, your original envelope is just as likely to have $2x$ dollars as it is to have x. But is that even possible? No, because if you think about any probability distribution over positive integers, it's going to have to taper off at some point as you get to larger numbers, so that larger amounts of money become less likely than smaller amounts.

2 Why Do We Need Randomness in Computer Science?

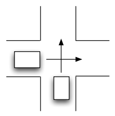
Why is randomness important in computer science? Let's take a look at some examples where randomness seems indispensable—or at the very least, a great help.

2.1 Cryptography

Randomness is absolutely essential to cryptography. Imagine if choosing an encryption key was based on a completely deterministic algorithm! If an eavesdropper can predict exactly what you'll do, then you have a problem.

2.2 Symmetry breaking

Randomness is useful for breaking up the "hallway dance", where you walk towards someone, and you both move to the same side to get out of each other's way, then both to the other side, then both back to the original side, in a continual dance.



Imagine two robot cars at an intersection. If both cars are programmed the same way—and if furthermore they can't distinguish left from right—then they won't be able to move because they'll both be doing the same thing, which would mean entering the intersection at the same time and crashing. On the other hand, if the algorithm governing the cars has a random component, both cars could have a chance of making it through the intersection unscathed. The fascinating field of *distributed computing* (which we unfortunately won't have time to delve into in this course) is full of examples like this. For many distributed computing problems of practical interest, it's possible to prove that there's no solution if all of the agents behave deterministically.

2.3 Database checks

Given two databases, each with 1 terabyte of information, suppose we want to check to see if the contents of the databases are identical. Additionally, suppose the databases are on different continents, so we need to send information over the internet to compare them. How can we do this? Of course, we could just send one database over the internet, but even with today's broadband rates, sending a terabyte probably isn't practical.

Can we just pick a random index and compare that index for both databases? What if the databases differ in just one place? How about summing the contents and comparing the sums? Better, but the sum of the contents of the first database, $x_1 + \cdots + x_n$, could be equal to the sum of the contents of the second database, $y_1 + \cdots + y_n$, even though the actual contents are different.

Comparing checksums is the right idea, but we need to add randomness to make it work. One approach is to interpret the databases as giant integers written in binary:

$$X = x_1 + 2x_2 + 4x_3 + 8x_4 + ...$$

$$Y = y_1 + 2y_2 + 4y_3 + 8y_4 + ...$$

Next, we pick some small random prime number, p. Then we check whether

$$X \bmod p = Y \bmod p.$$

We know that if two numbers are equal, then their remainder modulo anything is also equal. But if two numbers are *not* equal, and we pick a small prime at random and look at the two numbers modulo that random prime, then can we say that with high probability the results will be different?

Incidentally, here's a crucial point about randomized computation: you can never make the assumption that the inputs (in this case the databases) are random. You don't know where an input came from; it's just something someone gave you. The only assumption you can make is that the bits that you specifically *picked* to be random are random.

The question here is: $X - Y \overset{?}{=} 0 \bmod p$. This equality holds exactly when p divides $X - Y$. So now the question is how many different prime numbers can divide $X - Y$? Say $X - Y$ is an n-bit number. Then at most n different primes can divide it, since every prime number is at least 2. On the other hand, by the famous Prime Number Theorem, there are at least $\sim \frac{M}{\ln M}$ primes up to a given number M. So let's say we pick a random prime p with at most $10 \log n$ bits. Then p will be at most n^{10}. The number of primes less than n^{10} is quite large (roughly $\frac{n^{10}}{10 \ln n}$), but the number of possible divisors that $X - Y$ could have is at most n. Therefore, when we pick a random prime, with very high likelihood, we are going to find one that does not divide $X - Y$.

Getting back to our original problem: if we are comparing the contents of two n-bit databases, if we just look at them as integers modulo a random $O(\log n)$-bit prime, then if the two databases are different anywhere, we will see that difference with extremely high probability. This is called **fingerprinting**.

2.4 Monte Carlo simulation

Imagine you are trying to simulate a physical system and you want to get a feel for the aggregate behavior of the system. However, you don't have time to simulate every possible set of starting conditions. By picking various starting conditions at random, you can instead do a random sampling (also called *Monte Carlo simulation*).

But here we already encounter a fundamental question. Do you *really* need random starting conditions? Or would "pseudorandom" starting conditions work just as well? Here pseudorandom numbers are numbers that are meant to *look* random, but that are actually generated by (perhaps complicated) deterministic rules. These rules might initially be "seeded" by something presumed to be random—for example, the current system time, or the last digit of the Dow Jones average, or even the user banging away randomly on the keyboard. But from then on, all further numbers are generated deterministically.

In practice, almost all computer programs (including Monte Carlo simulations) use pseudorandomness instead of the real thing. The danger, of course, is that even if pseudorandom numbers *look* random, they might have hidden regularities that matter for the intended application but just haven't been noticed yet.

2.5 Polynomial equality

Given two polynomials of degree d, each of which is described by some complicated arithmetic formula, suppose we want to test whether they are equal or not.

$$(1-x)^4(x^2-3)^{17} - (x-5)^{62} \stackrel{?}{=} (x-4)^8(x^2+2x)^{700}$$

Sure, we could just expand both polynomials and check whether the coefficients match, but that could take exponential time. Can we do better?

It turns out that we can: we can simply pick random values for the argument x, plug them in, and see if the polynomials evaluate to the same value. If any of the values result in an inequality, then the polynomials are not equal.

Just like in the fingerprinting scenario, the key question is this: if two polynomials are different, then for how many different values of x can they evaluate to the same thing? The answer is d because of the **Fundamental Theorem of Algebra**.

1. Any non-constant polynomial has to have at least one root.

2. A non-constant polynomial of degree d has at most d roots.

When trying to test if two polynomials are equal, we are just trying to determine if their difference is equal to zero or not. As long as x is being chosen at random from a field which is much larger than the degree of the polynomial, chances are if we plug in a random x, the polynomial difference will *not* evaluate to zero, because otherwise the Fundamental Theorem of Algebra would be violated. So, we know that if two polynomials are different, then with high probability they will differ at any random point where we choose to evaluate them. Interestingly, to this day we don't know how to pick points where the polynomials are different in a deterministic way. We just know how to do it randomly.

3 Basic Tools for Reasoning about Probability

It's time to get more formal about what we mean by probabilities.

A	an event
$Pr[A]$	the probability that event A happens
$Pr[\neg A] = 1 - Pr[A]$	obvious rule
$Pr[A \vee B] = Pr[A] + Pr[B] - Pr[A \wedge B]$	subtract $Pr[A \wedge B]$ so we don't double count
$\leq Pr[A] + Pr[B]$	**Union Bound**

The union bound is one of the most useful facts in computer science. It says that even if there are all sorts of bad things that could happen to you, if each *individual* bad thing has a small enough chance of happening, then overall you're OK. In computer science, there are usually many different things that could cause an algorithm to fail, which are correlated with each other in nasty and poorly-understood ways. But the union bound says we can upper-bound the probability of *any* bad thing happening, by the sum of the probabilities of each *individual* bad thing—completely ignoring the complicated correlations between one bad thing and the next.

$Pr[A \wedge B] = Pr[A]Pr[B]$	if and only if A and B are independent
$Pr[A\|B] = \frac{Pr[A \wedge B]}{Pr[B]}$	definition of conditional probability
$= \frac{Pr[B\|A]Pr[A]}{Pr[B]}$	**Bayes' Rule**

Bayes' Rule comes up often when calculating the probability of some hypothesis being true given the evidence that you've seen. To prove Bayes' Rule, we just need to multiply both sides by $Pr[B]$, to get $Pr[A|B]Pr[B] = Pr[B|A]Pr[A] = Pr[A \wedge B]$.

We'll also need the concepts of *random variables* and *expectations.*

x	a random variable
$Ex[X] = \sum_i Pr[X = i]i$	expectation of X
$Ex[X + Y] = Ex[X] + Ex[Y]$	linearity of expectation
$Ex[XY] = Ex[X]Ex[Y]$	only if X and Y are independent