Lecture 13: Randomization

Anup Rao

February 16, 2023

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 × n-matrices A, B, C
Result: Whether or not A · B = C.
Sample an n coordiante column vector r ∈ {0,1}^{0,1} uniformly at
  random ;
if A(B(r)) = 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 $r_j = (1/d_{ij}) \sum_{k \neq j} d_{ik} r_k$. Define the random variable $A = (1/d_{ij}) \sum_{k \neq j} d_{ik} r_k$, and observe

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

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 φ
Result: A satisfying assignment for φ if one exists
Set a = 0 to be the n-bit all 0 string;
for i = 1, 2, ..., 100n^2 do
    if φ(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 ϕ is not satisfiable, then clearly the algorithm has a correct output. Now suppose ϕ 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

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

Then we have the following equations:

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

Thus we can compute:

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

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

Lemma 1.

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

Proof We have that

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

Therefore,

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

■

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 $x \mod p$ to Bob. Bob checks that $x \mod p$ is equal to $y \mod p$. 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 2 (Prime number theorem)**.** *Let $\pi(a)$ denote the number of primes that are at most a. Then* $\lim_{a\to\infty} \frac{\pi(a)}{a/\ln a} = 1$.

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 $O(1/c)$.