Randomized Algorithms and Probabilistic Techniques

Khalid Hourani

September 17, 2021

1 08/24

This course will involve only a few techniques with a variety of applications. Many domains, such as Distributed Computing, Networks, etc., all depend on randomized algorithms.

We will focus on a few techniques that lead to understanding.

1. Union Bound
2. Linearity of Expectation
3. Markov's Inequality
4. Chernoff Bounds

These four things prove very useful in the design and understanding of algorithms.

1.1 Basic Definitions

Definition 1.1

A *Sample Space* is a set S whose elements consist of *simple events* (also called *elementary* events). When S is finite or countably infinite, we say it is a *Discrete* sample space.

Definition 1.2

An *event* in a sample space S is a subset of S.

Definition 1.3

A *Probability Distribution* on S is a function $\mathbb{P} : 2^S \to [0, 1]$ that satisfies

1. $\mathbb{P}(S) = 1$
2. If E_1, E_2, ... are pairwise disjoint events (i.e., $E_i \cap E_j = \emptyset$ for all pairs i, j, also called mutually exclusive), ed by some finite or countably infinite set I, then

$$\mathbb{P}\left(\bigcup_{i \in I} E_i\right) = \sum_{i \in I} \mathbb{P}(E_i)$$

1.2 Conditional Probability

The expression $\mathbb{P}(E_2 \mid E_1)$ is read "the probability of E_2 given E_1." For example, if we randomly select a person from Texas, we might write

$$E_1 = \text{person chosen is in Houston}$$
$$E_2 = \text{person chosen is a UH student}$$

in which case $\mathbb{P}(E_2 \mid E_1)$ is simply the probability that a randomly selected person from Texas is a UH student *given that they are in Houston*.

Definition 1.4 ▶ Conditional Probability

The conditional probability $\mathbb{P}(E_2 \mid E_1)$ is defined

$$\mathbb{P}(E_2 \mid E_1) = \frac{\mathbb{P}(E_2 \cap E_1)}{\mathbb{P}(E_1)}$$

Intuitively, this can be thought of as taking the probability that E_2 and E_1 occur and "normalizing it" by dividing by the probability that E_1 occurs.

1.3 Independence

Definition 1.5 ▶ Independent Events

Two events, E_1 and E_2, are *independent* if

$$\mathbb{P}(E_1 \cap E_2) = \mathbb{P}(E_1)\,\mathbb{P}(E_2)$$

or, equivalently, if

$$\mathbb{P}(E_1 \mid E_2) = \mathbb{P}(E_1)$$

For example, suppose C_1 and C_2 are the outcomes of two fair coin tosses. These are independent, since

$$\mathbb{P}(C_1 = H \cap C_2 = T) = \mathbb{P}(C_1 = H)\,\mathbb{P}(C_2 = T) = \frac{1}{4}$$

Independence is not always related to physical independence. For example, say we are given a fair die and let

$$\begin{aligned} E_1 &= \text{roll is even} \\ E_2 &= \text{roll is less than or equal to } 4 \end{aligned}$$

In this case, we can enumerate the sample space and explicitly determine $\mathbb{P}(E_1)$, $\mathbb{P}(E_2)$, $\mathbb{P}(E_1 \cap E_2)$, and $\mathbb{P}(E_1)\,\mathbb{P}(E_2)$, to see if the events are independent:

$$\begin{aligned} E_1 &= \{2, 4, 6\} \\ E_2 &= \{1, 2, 3, 4\} \\ E_1 \cap E_2 &= \{2, 4\} \end{aligned}$$

Then

$$\begin{aligned} \mathbb{P}(E_1) &= \frac{3}{6} = \frac{1}{2} \\ \mathbb{P}(E_2) &= \frac{4}{6} = \frac{2}{3} \\ \mathbb{P}(E_1 \cap E_2) &= \frac{2}{6} = \frac{1}{3} \\ \mathbb{P}(E_1)\,\mathbb{P}(E_2) &= \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3} \end{aligned}$$

Thus, we see the events are independent. On the other hand, if E_2 is the event that the roll is *strictly less* than 4, we have

$$\begin{aligned} E_1 &= \{2, 4, 6\} \\ E_2 &= \{1, 2, 3\} \\ E_1 \cap E_2 &= \{2\} \end{aligned}$$

Then

$$\begin{aligned}\mathbb{P}(E_1) &= \frac{3}{6} = \frac{1}{2}\\ \mathbb{P}(E_2) &= \frac{3}{6} = \frac{1}{2}\\ \mathbb{P}(E_1 \cap E_2) &= \frac{1}{6}\\ \mathbb{P}(E_1)\,\mathbb{P}(E_2) &= \frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4}\end{aligned}$$

and we see that the events are *not independent*.

1.4 The Inclusion-Exclusion Principle

A basic result in set theory is that

$$\begin{aligned}|A \cup B| &= |A| + |B| - |A \cap B|\\ |A \cup B \cup C| &= |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|\end{aligned}$$

which can be generalized to an arbitrary finite union by

$$\bigcup_{i=1}^{n} A_i = \sum_{k=1}^{n} (-1)^{k+1} \left(\sum_{1 \le i_1 < \cdots < i_k \le n} |A_{i_1} \cap \cdots \cap A_{i_k}| \right)$$

or equivalently

$$\left| \bigcup_{i=1}^{n} A_i \right| = \sum_{\emptyset \neq J \subseteq \{1,\ldots,n\}} (-1)^{|J|+1} \left| \bigcap_{j \in J} A_j \right|$$

This yields the probability formulas

Theorem 1.1 ▶ Inclusion-Exclusion Principle

For any events E_1, E_2, …, E_n,

$$\begin{aligned}\mathbb{P}(E_1 \cup E_2) &= \mathbb{P}(E_1) + \mathbb{P}(E_2) - \mathbb{P}(E_1 \cap E_2)\\ \mathbb{P}\left(\bigcup_{i=1}^{n} E_i \right) &= \sum_{\emptyset \neq J \subseteq \{1,\ldots,n\}} (-1)^{|J|+1}\, \mathbb{P}\left(\bigcap_{j \in J} E_j \right)\end{aligned}$$

1.5 Union Bound

Theorem 1.2 ▶ Union Bound

Let E_1, E_2, …, be any countable set of events. Then

$$\mathbb{P}\Big(\bigcup E_i \Big) \le \sum \mathbb{P}(E_i)$$

While this bound is often not very precise, it is useful in many cases where the events E_i are "bad" and we can bound the likelihood of a single E_i. This allows us to bound the likelihood of *any* E_i.

1.6 Conditioning on Multiple Events (Chain Rule)

By repeatedly applying the definition of conditional probability, we have

$$\begin{aligned}\mathbb{P}(E_1 \cap E_2 \cap \cdots \cap E_n) &= \prod_{i=1}^{n} \mathbb{P}(E_i \mid E_1 \cap E_2 \cap \cdots \cap E_{i-1})\\ &= \mathbb{P}(E_1)\,\mathbb{P}(E_2 \mid E_1) \cdots \mathbb{P}(E_n \mid E_1 \cap E_2 \cap \cdots \cap E_{n-1})\end{aligned}$$

1.7 Birthday Paradox

Problem 1.1 ▶ Birthday Paradox

Suppose n people are in a room and they have birthdays chosen uniformly at random from the 365 calendar days. What is the probability that two people share a birthday?

The probability that *no two* people share a birthday can be calculated by considering the following events:

$$\begin{aligned} E_1 &= \text{person 1 has a birthday} \\ E_2 &= \text{person 2 has a different birthday than person 1} \\ &\vdots \\ E_i &= \text{person } i \text{ has a different birthday than people 1 through } i-1 \end{aligned}$$

Specifically, the probability that no two people share a birthday is

$$\begin{aligned} \mathbb{P}(E_1 \cap E_2 \cap \cdots \cap E_n) &= \mathbb{P}(E_1)\,\mathbb{P}(E_2 \mid E_1) \cdots \mathbb{P}(E_n \mid E_1 \cap E_2 \cap \cdots \cap E_{n-1}) \\ &= 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \cdots \times \left(1 - \frac{365-n+1}{365}\right) \end{aligned}$$

For what value of n is the above probability less than $1/2$?

2 08/26

2.1 Birthday Paradox

The probability that no two people share a birthday is

$$\begin{aligned} \mathbb{P}(E_1 \cap E_2 \cap \cdots \cap E_n) &= \mathbb{P}(E_1)\,\mathbb{P}(E_2 \mid E_1) \cdots \mathbb{P}(E_n \mid E_1 \cap E_2 \cap \cdots \cap E_{n-1}) \\ &= 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \cdots \times \left(1 - \frac{365-n+1}{365}\right) \end{aligned}$$

Applying the inequality $1 - x \leq e^{-x}$ when $|x| < 1$

$$\begin{aligned} &\leq 1 \cdot e^{-\frac{1}{365}} \cdot e^{-\frac{2}{365}} \cdots e^{-\frac{n-1}{365}} \\ &= e^{-\frac{n(n-1)}{2 \cdot 365}} \end{aligned}$$

Setting

$$e^{-\frac{n(n-1)}{2 \cdot 365}} \leq \frac{1}{2}$$

yields $n \geq 23$.

2.2 Randomized Algorithms

Problem 2.1 ▶ Toy Problem

Let A be an array of n numbers with the property that at least one number occurs at least $n/2$ times and the rest are distinct. Our goal is to find the element with duplicate entries.

Any deterministic algorithm will require at least $n/2 + 1$ operations. However, we can construct a randomized algorithm that only requires $\mathcal{O}(\log n)$ operations.

We do this by *random sampling*. Assume that choosing a single element in $\{a_1, a_2, \ldots, a_n\}$ requires $\mathcal{O}(1)$ time and that we sample *with replacement*. We will frequently use sampling with replacement because it allows the samples to be independent, and therefore simplifies the analysis. Now, consider the following algorithm:

Algorithm 1 Randomized Algorithm for Toy Problem

```
Function Rand-Duplicate-Toy(arr):
    loop:
        sample_1 ← Sample(arr)
        sample_2 ← Sample(arr)
        if sample_1 ≠ sample_2:
            return sample_1
```

This algorithm simply chooses two elements (not necessarily distinct) at random from the array and repeats until the two elements are identical, after which it outputs the sampled element as the duplicate.

Let us analyze the probability that this algorithm outputs the duplicate value after a single iteration. Suppose the two values chosen are a_i and a_j. Then

$$\begin{aligned}\mathbb{P}(a_i = a_j \text{ and } i \neq j) &= \frac{n/2}{n} \cdot \frac{n/2-1}{n} \\ &= \frac{1}{2}\left(\frac{1}{2} - \frac{1}{n}\right) \\ &= \frac{1}{4} - \frac{1}{2n} \\ &\geq \frac{3}{2} \text{ for } n \geq 5\end{aligned}$$

What is the probability, then, that our algorithm fails after k iterations of Algorithm 1? These events are independent, hence the probability is bounded above by

$$\begin{aligned}\mathbb{P}(\text{failure}) &\leq \left(1 - \frac{3}{20}\right)^k \\ &\leq e^{-\frac{3}{20}k}\end{aligned}$$

Setting $k = \frac{20}{3}\ln n$,

$$\begin{aligned}&= e^{-\ln n} \\ &= \frac{1}{n}\end{aligned}$$

2.3 Baye's Law

Theorem 2.1 ▶ Baye's Law

For any events A and B,

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(B \mid A)\,\mathbb{P}(A)}{\mathbb{P}(B)}$$

2.4 Concepts in Randomized Algorithms

Definition 2.1

An algorithm is *Monte Carlo* if it has a non-zero probability of outputting an incorrect solution. If an algorithm will output the correct solution with probability 1, it is *Las Vegas*.

Definition 2.2 ▶ High Probability

Given an input of size n, we say an event occurs *with high probability* (whp) if it occurs with probability $1 - 1/n^c$ for some $c > 0$.

Problem 2.2 ▶ Modified Toy Problem

Let A be an array of n numbers with the property that *either* all elements are distinct *or* $n/2$ elements are repeated.

Let us define

$$E_1 = \text{event that exactly } n/2 \text{ elements are distinct}$$
$$E_2 = \text{event that all } n \text{ elements are distinct}$$

At first, our "belief" in E_1 and E_2 are approximately the same, i.e., we have no information to determine which event is more likely. However, let us assume we have sampled two elements with replacement and determined that they are not equal (call this event F). We can update our "belief" via Baye's Law.

$$\begin{aligned}\mathbb{P}(E_1 \mid F) &= \frac{\mathbb{P}(F \mid E_1)\,\mathbb{P}(E_1)}{\mathbb{P}(F)} \\ &\leq \frac{\left(1-\frac{3}{20}\right)\cdot\frac{1}{2}}{\left(1-\frac{3}{20}\right)\cdot\frac{1}{2}+\frac{1}{2}} \\ &= \frac{17}{37}\end{aligned}$$

Thus, we see that the probability has shifted from "50-50" to skewing *in favor* of the belief that E_2 has occurred.

2.5 Law of Total Probability

Theorem 2.2 ▶ Law of Total Probability

For any event A and any countable partition of the sample space B_1, B_2, ..., that is, the events B_i are pairwise disjoint and their union forms the entire sample space, we have

$$\mathbb{P}(A) = \sum_i \mathbb{P}(A \cap B_i) = \sum_i \mathbb{P}(A \mid B_i)\,\mathbb{P}(B_i)$$

2.6 Naive Bayesian Classifier

A Naive Bayesian Classifier is a classic example of how we can apply probabilistic techniques and Baye's Law in practice. Suppose we have a *feature set* e, a vector of boolean n boolean variable. These could be, for example, whether a certain keyword appears in an email in an attempt to detect spam. Then, assuming the independence of the boolean variables (this is why it is referred to as *naive*), we have

$$\begin{aligned}\mathbb{P}(E \mid e = (b_1, b_2, \ldots, b_n)) &= \frac{\mathbb{P}(e = (b_1, b_2, \ldots, b_n) \mid E)\,\mathbb{P}(E)}{\mathbb{P}(e = (b_1, b_2, \ldots, b_n))} \\ &= \frac{\mathbb{P}(b_1 \mid E)\cdot\mathbb{P}(b_2 \mid E)\cdots\mathbb{P}(b_n \mid E)\cdot\mathbb{P}(event)}{\mathbb{P}(e = (b_1, b_2, \ldots, b_n))}\end{aligned}$$

3 08/31

3.1 Verifying Matrix Multiplication

Problem 3.1 ▶ Verifying Matrix Multiplication

Let A, B, and C be $n \times n$ matrices. Verify whether

$$A \times B = C$$

The best known deterministic algorithm for this requires $\mathcal{O}(n^{2.37286})$ operations, improving over the previous best of $\mathcal{O}(n^{2.37287})$ [1]. However, we can do substantially better using randomization.

A principle in randomized algorithms is to take advantage of an "abundance of weaknesses," that is, when some property that is helpful towards your algorithm occurs often enough, to try and make use of that property.

In this case, the weakness is that, if $AB = C$, then $AB\mathbf{r} = C\mathbf{r}$ for every vector $\mathbf{r}$. Thus, a single vector will prove if $AB \neq C$, and each vector for which $AB\mathbf{r} = C\mathbf{r}$ increases our confidence that $AB = C$. This admits the following pseudocode Notice that this algorithm only requires $\mathcal{O}(n^2)$ operations. As stated

Algorithm 2 Verify Matrix Multiplication

```
Function RAND-MATRIX-MULT-VERIFY(A, B, C):
    r ← SAMPLE({0,1}^n)
    if A(Br) ≠ Cr:
        return False
    else:
        return True
```

previously, it does not admit false negatives — if $AB = C$, it will never output `False`, because we must have $AB\mathbf{r} = C\mathbf{r}$ for all $\mathbf{r}$.

Let us determine the probability that this algorithm outputs `True` when $AB \neq C$. Write $D = AB - C \neq 0_{n,n}$, and observe that D must have at least one non-zero entry. Without loss of generality, say it is $d_{1,1}$. Now, if $AB\mathbf{r} = C\mathbf{r}$, then $D\mathbf{r} = \mathbf{0}$. In particular, writing $\mathbf{r} = (r_1, r_2, \ldots, r_n)$, we must have

$$d_{1,1}r_1 + d_{1,2}r_2 + \cdots + d_{1,n}r_n = 0$$

$$r_1 = \frac{-1}{d_{1,1}} \sum_{i=2}^{n} d_{1,i}r_i$$

To determine the probability that r_1 is equal to this value, we apply the *Principle of Deferred Decisions*

Definition 3.1 ▶ Principle of Deferred Decisions

The *Principle of Deferred Decisions* states that, in the analysis of a randomized algorithm, we can *defer* random choices until they are revealed to the algorithm.

In this specific case, we can assume r_2, r_3, ..., r_n have already been assigned their random values. Clearly, then, the probability that r_1 is equal to $\frac{-1}{d_{1,1}} \sum_{i=2}^{n} d_{1,i}r_i$ is *at most* $1/2$.

This gives us an upper bound on the probability of a false positive. In particular, after repeating the algorithm k times, the probability of a false positive is at most $1/2^k$. Setting $k = \log n$ yields the desired upper bound of $1/n$. In other words, the following pseudocode verifies matrix multiplication with high probability

Algorithm 3 Verify Matrix Multiplication with High probability

```
Function RAND-MATRIX-MULT-VERIFY-WHP(A, B, C):
    for i ← 1 to log n:
        result ← RAND-MATRIX-MULT-VERIFY(A, B, C)
        if result is False:
            return False
    return True
```

3.2 Verifying Equality of Binary Strings

Problem 3.2 ▶ Verifying Equality of Binary Strings

Given strings

$$\mathtt{X} = \mathtt{x}_0\mathtt{x}_1 \dots \mathtt{x}_{n-1}$$
$$\mathtt{Y} = \mathtt{y}_0\mathtt{y}_1 \cdots \mathtt{y}_{n-1}$$

Determine if $X = Y$, i.e., if there exists an i such that $\mathtt{x}_i \neq \mathtt{y}_i$.

Here, we are interested in *communication complexity*. Specifically suppose Alice has string $\mathtt{X}$ and Bob has string $\mathtt{Y}$, and they wish to determine if their strings are equal by transmitting a message of k bits.

Clearly, there is a naive deterministic algorithm that requires $\mathcal{O}(n)$ time by simplying comparing all n bits. In fact, *every* deterministic algorithm requires $\Omega(n)$ bits. However, we can construct a randomized algorithm, via a technique called *fingerprinting*, that requires $\mathcal{O}(\log n)$ bits.

For a prime $p \in \{1, 2, \dots, k\}$, let us define $F_p(x) = x \bmod p$. Additionally, we will identify the values $\mathtt{X}$ and $\mathtt{Y}$ by the numerical value of their binary strings, that is,

$$\mathtt{X} = \sum_{i=0}^{n-1} \mathtt{x}_i 2^i$$
$$\mathtt{Y} = \sum_{i=0}^{n-1} \mathtt{y}_i 2^i$$

If $F_p(\mathtt{X}) \neq F_p(\mathtt{Y})$, then we output `False`, otherwise we output `True`. Again, this admits no false negatives, since if $\mathtt{X} = \mathtt{Y}$, $F_p(\mathtt{X}) = F_p(\mathtt{Y})$ for any function F_p.

Observe that, if $F_p(\mathtt{X}) = F_p(\mathtt{Y})$, then

$$\sum_{i=0}^{n-1} (\mathtt{x}_i - \mathtt{y}_i) 2^i \bmod p = 0$$

Additionally, notice that

$$\sum_{i=0}^{n-1} (\mathtt{x}_i - \mathtt{y}_i) 2^i \leq 2^n$$

In particular, there are at most n prime factors of $\sum_{i=0}^{n-1} (\mathtt{x}_i - \mathtt{y}_i) 2^i$. Letting $\pi(k)$ denote the number of primes less than or equal to k, we have

$$\begin{aligned} \mathbb{P}(F_p(\mathtt{X}) \neq F_p(\mathtt{Y}) \mid \mathtt{X} \neq \mathtt{Y}) &\leq \frac{n}{\pi(k)} \\ &\approx \frac{n \ln k}{k} \end{aligned}$$

Taking $k = n^2 \ln n$ yields

$$\mathbb{P}(F_p(\mathtt{X}) \neq F_p(\mathtt{Y}) \mid \mathtt{X} \neq \mathtt{Y}) \leq \frac{1}{n}$$

3.3 Binary Consensus

Problem 3.3 ▶ Binary Consensus

Given n parties, of which at most f are faulty (and can crash at any time), and who each hold a 0 or 1, Consensus is reached when:

1. Agreement — all non-faulty parties agree to the same bit
2. Validity — if all parties start with a value, all parties must agree on that value
3. Termination — all parties terminate in finite time

Let us assume that communication is asynchronous and that all nodes can communicate with all other nodes (i.e., the network is a complete graph).

With *no faults*, it is sufficient for each node to simply broadcast their value and take the majority, breaking ties with 1. However, with even a single fault, this problem *cannot* be solved deterministically. Additionally, no randomized algorithm can solve this problem when there are at least $n/2$ faults.

4 09/02

The Ben-Or algorithm can be used to solve Binary Consensus with faults.

Algorithm 4 Ben-Or Algorithm from view of processor p with initial value χ_p

```
1:  procedure BEN-OR
2:      decided ← False
3:      round ← 1
4:      while not decided:
5:          BROADCAST(1, round, χ_p)
6:          wait for n − f messages of type (1, round, ·)
7:          if there are more than n/2 messages of type (1, round, v):
8:              BROADCAST(2, round, D, v)
9:          else:
10:             BROADCAST(2, round, U)
11:         wait for n − f messages of type (2, round, ·)
12:         t ← number of messages of type (2, round, D, v)
13:         if t > 0:
14:             χ_p ← v
15:             if t > f:
16:                 decided ← True
17:                 BROADCAST(1, round, χ_p)
18:         else:                          ▷ All messages are of type (2, round, U)
19:             χ_p ← SAMPLE({0, 1})
20:         round ← round + 1
```

Theorem 4.1

The above algorithm is correct.

Proof. Validity is clear: if all nodes start with v, then we skip to □

Theorem 4.2

The expected number of rounds of Ben-Or is 2^{n-1}. Additionally, with high probability, the number of rounds is less than $2^{n-1} \ln n$.

References

[1] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication, 2020.