Randomized Algorithms and Probabilistic Techniques

Khalid Hourani

September 20, 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

```
1: Function RAND-DUPLICATE-TOY(arr):
2:     loop:
3:         sample_1 ← SAMPLE(arr)
4:         sample_2 ← SAMPLE(arr)
5:         if sample_1 ≠ sample_2:
6:             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 line 2? 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 $2^{n-1} \ln n$.

Proof. If all nodes are of type $(2, \texttt{round}, ?)$, then with probability $2/2^n = 1/2^{n-1}$ all nodes will choose the same value. Thus, we can upper bound the probability of failure after k rounds by

$$\mathbb{P}(\text{failure}) \leq \left(1 - \frac{1}{2^{n-1}}\right)^k \leq e^{\frac{-k}{2^{n-1}}}$$

setting $k = 2^{n-1} \ln n$

$$= \frac{1}{n}$$

Thus, the probability of success after $2^{n-1} \ln n$ rounds is at least $1 - 1/n$. Additionally, the expected number of rounds is at most 2^{n-1}. □

4.1 Graphs

Problem 4.1 ▶ Min-Cut

Given a graph $G = (V, E)$, find a minimum set of *edges* whose removal disconnects the graph.

Definition 4.1 ▶ Graph Cut

A *cut* is a partition of the vertices into two non-empty subsets, A and B.

Notice that any edge from A to B crosses the cut, and when all such edges are removed, the graph becomes disconnected.

In general, there are $\binom{n}{2}$ possible min-cuts.

Algorithm 5 Karger's Algorithm

```
1: Procedure KARGER():
2:     for i ← 1 to n − 2:
3:         Choose a random edge e
4:         Contract edge e
5:     Output remaining edges
```

Observe that line 3 requires $\mathcal{O}(1)$ operations and line 4 requires $\mathcal{O}(n)$ operations, for a total of $\mathcal{O}(n^2)$. Additionally, a cut set in the contracted graph is a cut set in the original graph, thus we can only *increase* the size of the min-cut via line 4.

Theorem 4.3

Karger's Algorithm outputs a min-cut with probability at least $1/n^2$.

Proof. Suppose C is some min-cut in G and write $|C| = k$. In that case, every node must have degree at least k, and $m \geq nk/2$.

Let E_i denote the event that no edge in C is contracted in iteration i. Notice that the event that C is output is simply

$$\begin{aligned}
\mathbb{P}\left(\bigcap E_i\right) &= \mathbb{P}(E_1)\,\mathbb{P}(E_2 \mid E_1) \cdots \mathbb{P}(E_{n-1} \mid E_1 \cap E_2 \cap \cdots \cap E_{n-2}) \\
&= \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(\frac{2}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&= \frac{1}{\binom{n}{2}} \\
&\geq \frac{1}{n^2}
\end{aligned}$$

□

Thus, we can repeat this process $n^2 \ln n$ times to achieve a high probability that the algorithm outputs a min-cut.

5 09/07

Definition 5.1 ▶ Random Variable

A *random variable* is a function from some sample space S to a set of numbers.

Definition 5.2 ▶ Probability of a Random Variable

Let $X : \Omega \to E$ be some random variable and let $S \subseteq E$. Then

$$\mathbb{P}(X \in S) = \mathbb{P}(\omega \in \Omega \mid X(\omega) \in S)$$

If S is a discrete probability space, then

$$\mathbb{P}(X = r) = \sum_{X(\omega)=r} \mathbb{P}(\omega)$$

For example, let us roll a fair die and say, should you roll i, you receive i^2 dollars. We can define the random variable X to be the money received in this dice game. Then

$$\mathbb{P}(X = 25) = \mathbb{P}(\text{roll} = 5) = \frac{1}{6}$$

On the other hand, say you receive i dollars (that is, the face of the die roll). Then

$$\mathbb{P}(X \equiv 0 \bmod 2) = \mathbb{P}(\text{die} \in \{2, 4, 6\}) = \frac{1}{2}$$

Suppose we additionally roll a second die and call the random variable equal to the face-value of this die Y. What is $\mathbb{P}(X + Y = 7)$? Say $R(i, j)$ is the event that the first die rolls i and the second rolls j. Then

$$\begin{aligned}
\mathbb{P}(X + Y = 7) &= \mathbb{P}(R(1,6) \cup R(2,5) \cup R(3,4) \cup R(4,3) \cup R(5,2) \cup R(6,1)) \\
&= \mathbb{P}(R(1,6)) + \mathbb{P}(R(2,5)) + \mathbb{P}(R(3,4)) + \mathbb{P}(R(4,3)) + \mathbb{P}(R(5,2)) + \mathbb{P}(R(6,1)) \\
&= \frac{6}{36} \\
&= \frac{1}{6}
\end{aligned}$$

5.1 Expected Value

Definition 5.3 ▶ Expected Value

Let X be a discrete random variable on a sample space S. The *expected value* (also called mean or average) of X is defined

$$\mathbb{E}[X] = \sum_{r \in X(S)} r\, \mathbb{P}(X = r)$$

For example, if X is the face-value of a fair die, then

$$\mathbb{E}[X] = \sum_{i=1}^{6} i\, \mathbb{P}(X = i) = 1 \cdot \frac{1}{6} + 1 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6} = \frac{7}{2}$$

Notice that $7/2$ is *not* an event in our sample space. In general, the expectation does not need to be a value that can be "achieved." Instead, the Law of Large Numbers formalizes the idea of what expectation is.

Theorem 5.1 ▶ Law of Large Numbers

Let X_1, X_2, ..., X_n denote the outcomes of n independent experiments, and suppose that experiment has expected value μ. Let $\overline{X}_n$ denote our *sample mean*, i.e., write

$$\overline{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

Then

$$\lim_{n \to \infty} \mathbb{P}\big(\overline{X}_n = \mu\big) = 1$$

Consider the following game: flip two fair coins. If both come up heads, win \$2, otherwise lose \$1. Let X denote the amount of money the player wins. Then

$$\mathbb{E}[X] = 2 \cdot \frac{1}{4} + -1 \cdot \frac{3}{4} = \frac{-1}{4}$$

We can define independence for random variables in the same way we did for events.

Definition 5.4 ▶ Independent Random Variables

Two random variables X and Y on a sample space S are independent if

$$\mathbb{P}(X = x \cap Y = y) = \mathbb{P}(X = x)\,\mathbb{P}(Y = y)$$

for all $x \in X(S)$ and $y \in Y(S)$

5.2 Linearity of Expectation

The following theorem proves very useful in analyzing randomized algorithms.

Theorem 5.2 ▶ Linearity of Expectation

For any random variables X and Y and real value c

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$
$$\mathbb{E}[cX] = c\,\mathbb{E}[X]$$

Proof. By definition

$$\begin{aligned}
\mathbb{E}[X+Y] &= \sum_{x \in X(S)} \sum_{y \in Y(S)} (x+y)\,\mathbb{P}(X = x \cap Y = y) \\
&= \sum_{x \in X(S)} \sum_{y \in Y(S)} x\,\mathbb{P}(X = x \cap Y = y) + \sum_{x \in X(S)} \sum_{y \in Y(S)} y\,\mathbb{P}(X = x \cap Y = y) \\
&= \sum_{x \in X(S)} \sum_{y \in Y(S)} x\,\mathbb{P}(X = x \cap Y = y) + \sum_{y \in Y(S)} \sum_{x \in X(S)} y\,\mathbb{P}(X = x \cap Y = y) \\
&= \sum_{x \in X(S)} x \sum_{y \in Y(S)} \mathbb{P}(X = x \cap Y = y) + \sum_{y \in Y(S)} y \sum_{x \in X(S)} \mathbb{P}(X = x \cap Y = y) \\
&= \sum_{x \in X(S)} x\,\mathbb{P}(X = x) + \sum_{y \in Y(S)} y\,\mathbb{P}(Y = y) \text{ by the law of total probability} \\
&= \mathbb{E}[X] + \mathbb{E}[Y]
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathbb{E}[cX] &= \sum_{x \in X(S)} cx\,\mathbb{P}(X = x) \\
&= c \sum_{x \in X(S)} x\,\mathbb{P}(X = x) \\
&= c\,\mathbb{E}[X]
\end{aligned}$$

□

The power of Linearity of Expectation can be seen by considering the following scenario: suppose we flip n fair coins and wish to count the expected number of heads, X. Without Theorem 5.2, this is

$$\mathbb{E}[X] = \sum_{i=0}^{n} i \binom{n}{i} \left(\frac{1}{2}\right)^n$$

a computationally difficult expression. On the other hand, if we let X_i denote the following random variable

$$X_i = \begin{cases} 1 & \text{coin } i \text{ lands heads} \\ 0 & \text{otherwise} \end{cases}$$

It should be quite clear that

$$X = \sum_{i=1}^{n} X_i$$

Now,

$$\begin{aligned} \mathbb{E}[X_i] &= 1\,\mathbb{P}(X_i = 1) + 0\,\mathbb{P}(X_i = 0) \\ &= \frac{1}{2} \end{aligned}$$

hence

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \frac{n}{2}$$

an almost *trivial* computation.

The above random variable X_i is called a Bernoulli or *indicator* random variable, since it *indicates* whether the desired outcome occurred. In general, if X is an indicator random variable, then

$$\mathbb{E}[X] = \mathbb{P}(X = 1)$$

We can similarly demonstrate the power of Linearity of Expectation with the following problem: suppose n people arrive at a party and drop off their coats, for which they receive a receipt. Unfortunately, the attendant loses all of the stubs and decides to assign the party-goers coats at random. What is the expected number of people who receive their original coat?

Let X denote the number of people who receive their own coats. Then

$$\mathbb{E}[X] = \sum_{i=0}^{n} i\,\mathbb{P}(X = i)$$

Notice that, for general i, this is impractical to determine. In fact, the number $D_{n,k}$ is called a *rencontres number*, and denote the number of permutations of $\{1, 2, \ldots, n\}$ with exactly k fixed points. This can be recursively computed with

$$D_{n,k} = \binom{n}{k} D_{n-k,0}$$

However, we can determine $\mathbb{E}[X]$ by applying Linearity of Expectation. Define

$$X_i = \begin{cases} 1 & \text{person } i \text{ receives their own coat} \\ 0 & \text{otherwise} \end{cases}$$

and observe that $X = \sum X_i$. Additionally,

$$\mathbb{E}[X_i] = \mathbb{P}(X_i = 1) = \frac{(n-1)!}{n!} = \frac{1}{n}$$

from which it follows that

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = n\frac{1}{n} = 1$$

5.3 3-SAT

Problem 5.1 ▶ Boolean Satisfiability

Given a boolean formula, determine if there exists an assignment of the variables that causes the formula to evaluate to `True`.

A formula is in *Conjunctive Normal Form* (CNF) if it can be written as a disjunction of conjunctions, that is, a formula B is in CNF if

$$B = T_1 \wedge T_2 \wedge \cdots \wedge T_n$$

and each T_i is of the form $t_1 \vee t_2 \vee \cdots \vee t_i$. We say a formula is k-CNF if each T_i consists of exactly k conjunctions.

It turns out that this problem is equivalent to the *3-SAT* problem.

Theorem 5.3

Given a boolean formula B, there is an *equisatisfiable* formula B' in 3CNF whose length is at most 3 times that of B. That is, the formula B is satisfiable if and only if B' is satisfiable.

Proof. Consider some clause of conjunctions $l_1 \vee l_2 \vee \cdots \vee l_n$. The formula

$$\begin{aligned}
&(l_1 \vee l_2 \vee x_2) \\
\wedge&(\neg x_2 \vee l_3 \vee x_3) \\
\wedge&(\neg x_3 \vee l_4 \vee x_4) \\
&\vdots \\
\wedge&(\neg x_{n-3} \vee l_{n-2} \vee x_{n-2}) \\
\wedge&(\neg x_{n-2} \vee l_{n-1} \vee l_n)
\end{aligned}$$

is equisatisfiable to the clause. □

Problem 5.2 ▶ Max 3-SAT

Given a 3-SAT problem, find an assignment of the variables that *maximizes* the number of clauses that evaluate to `True`.

Consider an algorithm where we simply assign each variable 0 or 1 at random. For clause i, define

$$X_i = \begin{cases} 1 & \text{clause } i \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

And set $X = \sum X_i$ to be the number of satisfied clauses. Suppose clause i is $a \vee b \vee c$. Then

$$\begin{aligned}
\mathbb{E}[X_i] &= \mathbb{P}(a \vee b \vee c) \\
&= 1 - \mathbb{P}(\neg a \wedge \neg b \wedge \neg c) \\
&= 1 - \left(\frac{1}{2}\right)^3 \\
&= \frac{7}{8}
\end{aligned}$$

Thus

$$\mathbb{E}[X] = \sum_{i=1}^{m} \mathbb{E}[X_i] = \frac{7m}{8}$$

5.4 Probabilistic Method

The *Probabilistic Method* is a nonconstructive method to demonstrate the existence of some sort of mathematical object. This is done by creating a probability distribution on some sample space and showing that such an object exists with probability greater than 0, from which you conclude that at least one such object exists.

Theorem 5.4

There is at least one assignment of variables that has $7m/8$ true clauses.

To see this, simply note that if $\mathbb{E}[X] = \mu$, there must be at least one value in the sample space whose value is greater than or equal to μ and at least one whose value is less than or equal to μ.

6 09/09

We can apply the algorithm given previously to actually *find* an assignment of the boolean variables which will satisfy at least $7m/8$ clauses. Specifically, consider the following algorithm Clearly, the above

Algorithm 6 Randomized Max 3-SAT Algorithm

```
Function SOLVE(formula):
    result ← 0
    while result < 7m/8:
        bool-vec, result ← TRY-SOLVE(formula)
    return bool-vec
Function TRY-SOLVE(formula):
    bool-vec ← SAMPLE({0,1}^n)
    k ← number of satisfied clauses in formula
    return (bool-vec, k)
```

algorithm will *eventually* return the desired boolean vector. However, we are concerned with how many iterations this algorithm will require. To determine this, we must first determine the probability that the randomly selected boolean vector will satisfy at least $7m/8$ clauses.

Let X represent the number of satisfied clauses. Recall that $\mathbb{E}[X] = 7m/8$. Let us call this value μ. Then

$$\begin{aligned}
\mu &= \mathbb{E}[X] \\
&= \sum_{x=0}^{m} x\,\mathbb{P}(X = x) \\
&= \sum_{x<\mu} x\,\mathbb{P}(X = x) + \sum_{x\geq\mu} x\,\mathbb{P}(X = x)
\end{aligned}$$

Notice that, since $x < \mu$, $x \leq \mu - c$ for some constant $0 < c \leq 1$

$$\leq (\mu - c)\sum_{x<\mu} \mathbb{P}(X = x) + m\sum_{x\geq\mu} \mathbb{P}(X = x)$$

setting $p = \mathbb{P}(X \geq \mu)$

$$= (\mu - c)(1 - p) + mp$$

Solving for p

$$\begin{aligned}
p &\geq \frac{c}{c - \mu + m} \\
&= \Theta\left(\frac{1}{m}\right)
\end{aligned}$$

Before we finish this analysis, we should consider the Geometric Random Variable.

6.1 Geometric Random Variable

Definition 6.1 ▶ Geometric Random Variable

Consider some Bernoulli trial, i.e., an event with exactly two possible outcomes, denotes "success" and "failure", and suppose success occurs with probability p. The *Geometric Distribution*, denoted $X \sim G(p)$, is the number of trials necessary until a success occurs.

Lemma 6.1

Let X be some discrete random variable that takes integer values. Then

$$\mathbb{E}[X] = \sum_{x=1} \mathbb{P}(X \geq x)$$

Proof.

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{x=1}^{\infty} x\, \mathbb{P}(X = x) \\
&= \sum_{x=1}^{\infty} \sum_{i=1}^{x} \mathbb{P}(X = x) \\
&= \sum_{x=1}^{\infty} \sum_{i=x}^{\infty} \mathbb{P}(X = i) \\
&= \sum_{x=1}^{\infty} \mathbb{P}(X \geq x)
\end{aligned}$$

□

Theorem 6.2 ▶ Expectation of Geometric Random Varaible

Let $X \sim G(p)$. Then $\mathbb{E}[X] = \frac{1}{p}$.

Proof. Observe that $\mathbb{P}(X = k) = (1-p)^{k-1}p$, hence

$$\mathbb{P}(X \geq k) = (1-p)^{k-1}$$

and

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{k=1}^{\infty} k\, \mathbb{P}(X = k) \\
&= \sum_{k=1}^{\infty} \mathbb{P}(X \geq k) \\
&= \sum_{k=1}^{\infty} (1-p)^{i-1} \\
&= \frac{1}{1-(1-p)} \\
&= \frac{1}{p}
\end{aligned}$$

□

Now, returning to our Max 3-SAT algorithm, notice that the probability that it requires k iterations to output an assignment of at least $7m/8$ `True` clauses is

$$\begin{aligned}
(1-p)^k &\leq \left(1 - \frac{c}{m}\right)^k \text{ for some } c > 0 \\
&\leq e^{\frac{-ck}{m}}
\end{aligned}$$

setting $k = m/c \ln n$

$$= \frac{1}{m}$$

Observe that our algorithm is a Las Vegas algorithm. However, there is an analogous Monte Carlo algorithm:

Algorithm 7 Randomized Max 3-SAT Algorithm (MC Variant)

```
Function SOLVE(formula):
    result ← 0
    bool-vec ← 0
    for i ← 1 to m/c ln n:
        new-bool-vec, new-result ← TRY-SOLVE(formula)
        if new-result > result:
            result ← new-result
            bool-vec ← new-bool-vec
    return bool-vec
```

6.2 Binomial Random Variable

Definition 6.2 ▶ Geometric Random Variable

Consider some Bernoulli trial, i.e., an event with exactly two possible outcomes, denotes "success" and "failure", and suppose success occurs with probability p. The *Binomial Distribution*, denoted $X \sim B(n, p)$, is the number of successes after n trials.

Theorem 6.3

Let $X \sim B(n, p)$. Then $\mathbb{E}[X] = np$.

Proof. Again, Linearity of Expectation is useful here. Let X_i be the indicator random variable that is 1 when trial i is a success. Then

$$\mathbb{E}[X_i] = \mathbb{P}(X_i = 1) = p$$

hence

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = np$$

□

6.3 Randomized Quicksort

The *Quicksort* algorithm is a computer science staple, one of the most famous algorithms of the 20th century. A simplified version of the original implementation is below:

Algorithm 8 Simplified variant of Quicksort algorithm.

```
Function QUICKSORT(arr):
    n ← LEN(arr)
    if n ≤ 1 :
        return arr
    else:
        left, P, right ← PARTITION(arr)
        return QUICKSORT(left) + P + QUICKSORT(right)
Function PARTITION(arr):
    pivot ← arr[0]
    left, P, right ← [], [], []
    for a ∈ arr :
        if a < pivot :
            left.APPEND(a)
        else if a = pivot:
            P.APPEND(a)
        else:
            right.APPEND(a)
    return left, P, right
```

Notice that the worst case of this algorithm is $\mathcal{O}(n^2)$. On the other hand, the best case is $\mathcal{O}(n \log n)$. We can consider a randomized variant — functionally, the same algorithm, but with the partition scheme changed. Rather than selecting the element with index 0 as the pivot, select a *random* element as the pivot. Let us consider the number of comparisons in this variant.

Theorem 6.4

The above randomized variant of Quicksort has an expected runtime of $\mathcal{O}(n \log n)$.

Proof. Let X denote the number of comparisons. It should be clear that $\mathbb{P}(X = x)$ is impractical to compute directly. Instead, notice that two elements of the array will be compared *at most* once — when one of the two elements is the pivot *end*.

Assume the sorted order of the elements is a_0, a_1, ..., a_n, and let $X_{i,j}$ denote the indicator random variable that is 1 when a_i and a_j are compared. Notice that if any of a_{i+1}, a_{i+2}, ..., a_{j-1} are chosen, then a_i and a_j will end up in separate partitions and never be compared. Additionally, if any values a_0, a_1, ..., a_{i-1} or a_{j+1}, a_{j+2}, ..., a_{n-1} are chosen, this will have no effect on whether a_i and a_j are compared.

Thus, the probability that a_i and a_j are compared is simply the probability that either value is chosen as the pivot from a_i, a_{i+1}, ..., a_j, which is

$$\frac{2}{j-i+1}$$

Thus, we have

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \mathbb{E}[X_{i,j}] \\
&= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \frac{2}{j-i+1}
\end{aligned}$$

writing $k = j - i$

$$\begin{aligned}
&= \sum_{i=0}^{n-1} \sum_{k=1}^{n-i-1} \frac{2}{k+1} \\
&\leq \sum_{i=0}^{n-1} 2H_n \\
&\leq 2nH_n \\
&= \mathcal{O}(n \log n)
\end{aligned}$$

□

7 09/16

7.1 Randomized Quicksort

The following pseudocode demonstrates our randomized Quicksort variant.

Algorithm 9 Simplified variant of Quicksort algorithm with random pivot.

```
Function Rand-Quicksort(arr):
    n ← Len(arr)
    if n ≤ 1 :
        return arr
    else:
        left, P, right ← Partition(arr)
        return Rand-Quicksort(left) + P + Rand-Quicksort(right)
Function Partition(arr):
    pivot ← Sample(arr)
    left, P, right ← [], [], []
    for a ∈ arr :
        if a < pivot :
            left.Append(a)
        else if a = pivot:
            P.Append(a)
        else:
            right.Append(a)
    return left, P, right
```

Notice that the algorithm is identical to Algorithm 8, except for the partition choice. Later, we will apply Chernoff Bounds to show that, with high probability, the number of comparisons is $\mathcal{O}(n \log n)$.

Let us consider the original algorithm once more, wherein the pivot is always taken to be `arr`[0]. Suppose the input is a randomly selected *permutation* of the array. We can now determine the expected number of comparisons of this algorithm (this is commonly called *average case* or *probabilistic* analysis).

In randomized Quicksort, the analysis is with respect to the *random pivot choice*. On the other hand, here we are analyzing with respect to the *random input*.

However, the analysis is largely the same. Again, we let $X_{i,j}$ be the indicator random variable equal to 1 when elements a_i and a_j are compared. These elements will be compared when a_i occurs before a_j (or vice-versa) in the permutation, and none of a_{i+1}, a_{i+2}, ..., a_{j-1} are chosen first. Apply the principle of deferred decisions and assume all values, other than a_i, a_{i+1}, ..., a_j have been placed. There are $(j-i+1)!$ permutations of these values, of which $2(j-i-1)!$ have a_i or a_j first. Thus,

$$\mathbb{E}[X_{i,j}] = \frac{2(j-i-1)}{(j-i+1)} = \frac{2}{(j-i+1)}$$

from which the result follows.

7.2 Selection Algorithm

Problem 7.1 ▶ Selection

Given an array of length n, find the k-th smallest element of n.

We can apply an algorithm similar to Quicksort: select a pivot and partition the array. If the number of elements less than the pivot is $k-1$, then the pivot is the k-th smallest. Otherwise, the desired element falls in the left or right partition.

Algorithm 10 Simplified variant of Quickselect algorithm with random pivot.

```
1:  Function RAND-QUICKSELECT(arr):
2:      n ← LEN(arr)
3:      if n ≤ 1 :
4:          return arr
5:      else:
6:          left, P, right ← PARTITION(arr)
7:          ℓ ← LENGTH(left)
8:          p ← LENGTH(P)
9:          if ℓ ≥ k:                          ▷ k-th smallest is in left
10:             return RAND-QUICKSELECT(left, k)
11:         else if ℓ + p < k:                 ▷ k-th smallest is in right
12:             return RAND-QUICKSELECT(right, k − ℓ − p)
13:         else:
14:             return pivot
```

References

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