CS331: Algorithms and Complexity
Part VII: Randomized Algorithms

Kevin Tian

1 Introduction

Randomness is a powerful resource in the design of algorithms. For many fundamental algorithmic problems, the guarantees known to be achievable deterministically lag behind those achieved when granted access to randomness; this is certainly true for measures of complexity such as runtime, but extends to other resources such as space.[1] Moreover, randomized algorithms are often simpler and more elegant than deterministic counterparts, which is frequently beneficial in practice. Finally, in modern data science applications, the algorithmic problem itself is often inherently random, i.e., our goal is to learn about a distribution from random samples.

These notes introduce the design and analysis of *randomized algorithms*. While randomized solutions are often clean and intuitive, there are many subtleties that arise in their analysis, so algorithm designers must be careful. For instance, how do we quantify the performance of randomized algorithms, if they do not always succeed? We will explore several strategies for reasoning about a randomized algorithm $\mathcal{A}$'s "typical behavior," as well as statements of the form, with probability $1 - \delta$, $\mathcal{A}$ satisfies some guarantee. Here δ is called the "failure probability," i.e., the chance that $\mathcal{A}$ does not yield the desired outcome. In different applications of interest, different δ may be more appropriate; at minimum we would like δ to be a small constant (e.g., $\delta = 0.1$), but often, we can provide significantly stronger bounds (e.g., $\delta = \frac{1}{\text{poly}(n)}$).

As a basic example we have already seen, consider the Selection algorithm in Section 6.2, Part II. A key subroutine used was FindPivot(L), which takes as input a list L of n real numbers, and tries to find an entry $x \in L$ of "middling" rank. Here, the rank $\text{rank}(x)$ of an entry $x \in L$ is the number of entries in L that are no larger than x. In particular, our goal is to find the k^{th} largest entry, for any value k satisfying, say, $\frac{n}{4} \leq k \leq \frac{3n}{4}$. As a reminder, FindPivot is a core subroutine of both Selection and the famous Quicksort algorithm. Our solution in Section 6.2, Part II was based on the *median-of-medians* approach, and was quite complicated, interleaving Selection with FindPivot steps recursively. Recall that the payoff at the time was an $O(n)$ time FindPivot algorithm.

There is another, much simpler, solution that uses randomness to achieve the same runtime in expectation. First, observe that for any $x \in L$, we can compute $\text{rank}(L)$ in $O(n)$ time by maintaining a counter and comparing x to each other entry in a single pass. Call an entry $x \in L$ "middling" if $\frac{n}{4} \leq \text{rank}(x) \leq \frac{3n}{4}$. Specifically, consider selecting a uniformly random index $i \in [n]$, and returning $x \leftarrow L[i]$. Up to rounding errors, this algorithm succeeds with probability $\frac{1}{2}$ in returning a middling entry, and as mentioned, we can verify whether the algorithm succeeds in $O(n)$ time.

Our randomized implementation of FindPivot is now easy to describe: repeatedly sample an index $i \in [n]$ uniformly at random (called a "trial") until $\frac{n}{4} \leq \text{rank}(L[i]) \leq \frac{3n}{4}$. What is the expected number of trials we require until the algorithm finishes? We can perform a direct calculation: letting $X \in \mathbb{N}$ denote the random variable corresponding to the number of trials before termination. Then,

$$\Pr[X = i] = \left(\frac{1}{2}\right)^{i-1} \cdot \frac{1}{2} = \frac{1}{2^i} \text{ for all } i \in \mathbb{N},$$

[1] In some cases, this manifests as a polynomial-time speedup, i.e., a deterministic solution running in $O(n^2)$ time can be improved to $O(n)$ with randomness. However, there are certain problems for which there are simple, efficient randomized solutions, but it is not even known whether the problem is solvable deterministically in polynomial time. Up until somewhat recently, checking for primality of an integer was such an example, until the breakthrough deterministic algorithm of [AKS04]. Another example that still stands to this day is *polynomial identity testing*, i.e., testing whether a polynomial (given as an arithmetic circuit) always evaluates to zero.

because the algorithm must fail its first $i-1$ trials, and succeed on the last. We can then compute

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i\geq 1} \Pr[X=i]\cdot i = \sum_{i\geq 1}\frac{i}{2^i} = \frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\frac{4}{16}+\dots \\ &= \left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\dots\right)+\left(\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\dots\right)+\left(\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\dots\right)+\dots \\ &= 1+\frac{1}{2}+\frac{1}{4}+\dots = 2.\end{aligned} \tag{1}$$

Thus we are expected to succeed in just 2 trials, which amounts to a runtime of $2\cdot O(n) = O(n)$ when accounting for the time required for verification in each trial. We have thus matched the FindPivot algorithm's runtime from Section 6.2, Part II in expectation, but with a significantly more practical algorithm. This example is a first taste of both the simplicity and power of randomness, recurring themes that we explore throughout these notes.

2 Dependent random variables

One issue that frequently arises when randomness meets algorithm design is that the resulting random variables are often dependent. Recall that random variables X and Y are *independent* if the outcome of Y has no bearing on X's outcome, i.e., the conditional random variable $X \mid Y = y$ has the same distribution as X for any possible outcome y (for a refresher, see Section 6, Part I).

Independence is a convenient property for analysis purposes. Unfortunately, it is also completely at odds with how algorithms operate. Typical randomized algorithms make different decisions depending on the realization of random events, due to their iterative nature. For example, in our randomized FindPivot, whether or not the second trial occurs directly depends on the first trial's outcome. Such dependencies between iterations arise often in randomized algorithms.

In this section, we first explore two basic tools for reasoning about dependent random variables: *linearity of expectation* and the *union bound*, along with several motivating examples. We apply these tools to analyze a simple contention resolution scheme for distributed computing applications.

2.1 Basic tools

Here, we introduce two probabilistic techniques with wide general applicability. A major upside of both is that they apply equally well to independent and dependent random variables.

Linearity of expectation. Let X and Y be arbitrary real-valued random variables. The principle of linearity of expectation states that

$$\mathbb{E}[X+Y] = \mathbb{E}[X]+\mathbb{E}[Y]. \tag{2}$$

There is no catch here: (2) is literally always true. This is a ridiculously powerful fact, and shows up in the analysis of every randomized algorithm (e.g., as we need to bound the runtime of multiple steps). For instance, by recursively applying (2) we have the following generalization:

$$\mathbb{E}\left[\sum_{i\in[n]} X_i\right] = \sum_{i\in[n]}\mathbb{E}[X_i], \tag{3}$$

for any $n\in\mathbb{N}$ and real-valued random variables $\{X_i\}_{i\in[n]}$. Here are a few famous applications.

Suppose all n students in class give their backpack to a doorman, who writes down the names of each backpack's owner and memorizes everyone's face, matched to a name. If the doorman changes, the new doorman must randomly give backpacks back to students. How many students are expected to receive the correct backpack? There are many reasonable models here; we let the new doorman pick a uniformly random matching from names to students, and pass out backpacks based on it. Although this is a complicated distribution, the only fact we will need is that each student receives their backpack with probability $\frac{1}{n}$, which should be clear from symmetry.

Now we can solve the problem. Let X_i be the 0-1-valued random variable corresponding to whether student $i \in [n]$ received their backpack. If p is the probability that $X_i = 1$, and

$$\mathbb{E}[X_i] = p \cdot 1 + (1-p) \cdot 0 = p, \tag{4}$$

then we have shown $p = \frac{1}{n}$. This appears to be hard to relate to T, the total number of students who received the correct backpack, because T is an integer and each $\mathbb{E}[X_i] = p$ is fractional. However, $T = \sum_{i \in [n]} X_i$ by definition. Thus we can apply (3) and complete the calculation instantly:

$$\mathbb{E}[T] = \mathbb{E}\left[\sum_{i \in [n]} X_i\right] = \sum_{i \in [n]} \mathbb{E}[X_i] = n \cdot \frac{1}{n} = 1.$$

In this application, X_i was an example of an *indicator random variable*. More generally, for any random event $\mathcal{E}$ that may or may not happen, we define a corresponding indicator random variable $\mathbb{I}_{\mathcal{E}}$ that evaluates to 1 if $\mathcal{E}$ happens, and otherwise it evaluates to 0. This notation often simplifies proofs, as indicator random variables have many convenient interpretations: for example, a straightforward generalization of (4) shows that all events $\mathcal{E}$ satisfy

$$\mathbb{E}[\mathbb{I}_{\mathcal{E}}] = \Pr[\mathcal{E} \text{ occurs}]. \tag{5}$$

Next, we consider the *coupon collector* problem. In this problem, you are trying to collect n different coupons that arrive randomly in the mail every week, where you win a prize only if you assemble at least one copy of every coupon. All coupons you receive are uniformly random. What is the expected number of weeks that need to pass before you collect all unique coupons?

The basic observation is the following. Let $X_i \in \mathbb{N}$ be the number of weeks that pass after you receive your $(i-1)^{\text{th}}$ unique coupon, until you receive your i^{th} unique coupon. For example, if you receive your 4^{th} unique coupon the week after you receive your 3^{rd}, then $X_4 = 1$. We claim that

$$\mathbb{E}[X_i] = \frac{n}{n-(i-1)} \tag{6}$$

To see this, we need a more general principle. Suppose you have a two-sided coin that is biased to come up heads with probability $p \in (0,1)$. How many times do you have to toss it in expectation to see your first heads? Letting X be the number of tosses needed, X is an example of what is known as a *geometric random variable*. It is straightforward to check that

$$\Pr[X = i] = (1-p)^{i-1}p, \text{ for all } i \in \mathbb{N}.$$

Moreover, a generalization of (1) shows that

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i \geq 1} \Pr[X = i] \cdot i = \sum_{i \geq 1} i(1-p)^{i-1}p \\
&= \sum_{i \geq 1} (1-p)^{i-1}p + \sum_{i \geq 2} (1-p)^{i-1}p + \sum_{i \geq 3} (1-p)^{i-1}p \\
&= \sum_{i \geq 1} (1-p)^{i-1}p + (1-p)\sum_{i \geq 1} (1-p)^{i-1}p + (1-p)^2 \sum_{i \geq 1} (1-p)^{i-1}p + \ldots \\
&= \left(\sum_{j \geq 0} (1-p)^j\right)\left(\sum_{i \geq 1} (1-p)^{i-1}p\right) = \frac{1}{p} \cdot 1 = \frac{1}{p},
\end{aligned} \tag{7}$$

where we use that the sum of an infinite geometric sequence with first term a and common ratio $r \in (-1,1)$ is $\frac{a}{1-r}$. Now, returning to our coupon collector problem, X_i is a geometric random variable with mean $\Pr[X_i = 1] = \frac{n-(i-1)}{n}$, because in each week, there are $n-(i-1)$ new unique possible coupons, out of n in total. Thus applying (7) yields the claim (6).

Now we can conclude our calculation of $\mathbb{E}[\sum_{i \in [n]} X_i]$, the total number of weeks that must pass for all unique coupons to appear. We have by (6) that

$$\mathbb{E}\left[\sum_{i \in [n]} X_i\right] = \sum_{i \in [n]} \frac{n}{n-(i-1)} = n \cdot \sum_{i \in [n]} \frac{1}{i} = O(n \log(n)).$$

The last equality, $\sum_{i\in[n]} \frac{1}{i} = O(\log(n))$, is an excellent example of a continuous perspective making a discrete problem simpler: by bounding $\frac{1}{i} \le \int_{i-1}^{i} \frac{1}{t}\mathrm{d}t$, we have

$$\sum_{i\in[n]} \frac{1}{i} \le 1 + \sum_{i=2}^{n} \frac{1}{i} \le 1 + \int_{1}^{n-1} \frac{1}{t}\mathrm{d}t = 1 + \log(n-1) = O(\log(n)).$$

Union bound. The union bound is the second most general-purpose tool for reasoning about multiple, potentially dependent, random variables. Consider a set of k events, $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_k$ each of which may or may not happen. In particular, we allow for arbitrary dependencies between these events. However, one fact that is generically true is that

$$\Pr\left[\text{at least one of } \{\mathcal{E}_i\}_{i\in[k]} \text{ occurs}\right] \le \sum_{i\in[k]} \Pr[\mathcal{E}_i \text{ occurs}].$$

For shorthand, we will abbreviate the above expression by

$$\Pr\left[\bigcup_{i\in[k]} \mathcal{E}_i\right] \le \sum_{i\in[k]} \Pr\left[\mathcal{E}_i\right]. \tag{8}$$

The simplest proof of (8) uses the observation that for any events $\{\mathcal{E}_i\}_{i\in[k]}$,

$$\mathbb{I}_{\bigcup_{i\in[k]} \mathcal{E}_i} \le \sum_{i\in[k]} \mathbb{I}_{\mathcal{E}_i},$$

because the left-hand side is 0 unless at least one event occurs, in which case it is 1, but the right-hand side is clearly ≥ 1 in this case. By taking expectations of both sides, and applying (3) and (5), we have proven the union bound (8), without using any independence assumptions.

A classic application of (8) is the *birthday paradox*. Suppose each of the n students in class has a uniformly random birthday, all chosen independently out of m possible days of the year. What is the probability that no two students share a birthday? This question has real-life applications, e.g., in hashing, where we would like to prevent objects from randomly hashing to the same slot.

Alternatively, by taking complements, an equivalent question is: what is the probability that some pair $(i,j) \in [n]\times[n]$ with $i \ne j$ have the same birthday? For all such pairs (i,j), define an event $\mathcal{E}_{i,j}$ corresponding to students (i,j) sharing a birthday. Regardless of the i^{th} student's birthday, this probability is $\frac{1}{m}$ for a random independently sampled j^{th} student's birthday. Thus by (8),

$$\Pr\left[\bigcup_{\substack{(i,j)\in[n]\times[n]\\ i<j}} \mathcal{E}_{i,j}\right] \le \sum_{\substack{(i,j)\in[n]\times[n]\\ i<j}} \Pr\left[\mathcal{E}_{i,j}\right] = \binom{n}{2}\cdot\frac{1}{m} = \frac{n(n-1)}{2m}.$$

This shows that no two students share a birthday with probability at least $1 - \frac{n(n-1)}{2m}$. Thus, if e.g., $n \lesssim \sqrt{m}$, it is reasonably likely that no two students share a birthday. On the other hand, if $n \gtrsim \sqrt{m}$, then it is reasonably likely that a shared birthday (a.k.a. a "collision") occurs. The above calculation predicts that the transition from common collisions to uncommon collisions occurs around $n \approx \sqrt{2m}$. For example, if $m = 365$, then $\sqrt{2m} \approx 27$, which is surprisingly small.

This lower bound breaks down in the regime $n \gtrsim \sqrt{m}$, which requires stronger tools to understand. However, it turns out (though we will not prove it here) that the probability no two students share a birthday rapidly drops down to 0 once $n \gtrsim \sqrt{m}$. One provable heuristic to get a sense for the upper and lower tail behavior is that if $n \ge 3.1\sqrt{m}$, then a collision will occur with probability $\ge 90\%$, and conversely if $n \le 0.4\sqrt{m}$, then a collision will occur with probability $\le 10\%$.

2.2 Contention resolution

In this section, we apply the tools we have developed to a distributed computing problem, *contention resolution*. In this problem, we have $n \ge 2$ different processes which are competing for a

shared resource, e.g., a central server or database. Their goal is to each use the resource for one unit of time, so they can complete their job. However, the processes cannot communicate with each other, so at each time step, they can only make a binary decision on whether or not to attempt to access the shared resource. For example, the i^{th} machine can choose to make an attempt on time steps 1, 3, and 6, and wait on all other time steps. If it successfully becomes the only machine to request access to the shared resource on a given time step, it receives a message that it successfully completed its job. The goal is to design a protocol so that all n processes complete their jobs in as few time steps as possible. Clearly, the fewest time steps achievable is n.

This problem serves as a reasonable basic model of distributed computing, whether it be multiple processes on the same machine attempting to access a shared resource, or multiple servers in a network. It also is a first introduction to another computational resource beyond runtime that is often studied in distributed algorithmic settings: communication. In the variant of the contention resolution problem we consider, we model communication access in a fairly extreme way (i.e., no two processes can communicate), but more generally it can be interesting to bound distributed algorithm complexities under a communication budget.

We analyze the simplest possible protocol for this problem: at every time step, each process attempts to use the shared resource independently with probability $\frac{1}{n}$. Note that a process succeeds in being the unique process to access the shared resource in a time step, with probability

$$p := \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1},$$

because it needs to make an attempt (with probability $\frac{1}{n}$), and all of the other $n-1$ processes need to independently decide not to use the resource (with probability $1 - \frac{1}{n}$).

At this juncture, it is useful to mention the extremely useful approximation

$$\frac{1}{4} \leq \left(1 - \frac{1}{n}\right)^{n} \leq \frac{1}{e}, \text{ for all } n \geq 2. \tag{9}$$

The way to remember this approximation is that as $n \to \infty$, we know that $(1 - \frac{1}{n})^n \to \frac{1}{e}$, and this expression is minimized at $n = 2$, where we can check $(\frac{1}{2})^2 = \frac{1}{4}$. Therefore, we have

$$p = \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n} \geq \frac{1}{4n}.$$

We can thus bound the probability that a process $i \in [n]$ fails to uniquely access the shared resource in each of $T = \lceil 4Cn \log(n) \rceil$ time steps, for any constant $C > 1$, by

$$(1-p)^T \leq \left(1 - \frac{1}{4n}\right)^{T} \leq \left(\left(1 - \frac{1}{4n}\right)^{4n}\right)^{C \log(n)} \leq \left(\frac{1}{e}\right)^{C \log(n)} = \frac{1}{n^C}.$$

In the third inequality, we used the upper bound in (9) again. For example, the probability that process $i \in [n]$ fails to uniquely access the shared resource in each of $T > 8n \log(n)$ time steps is $\leq \frac{1}{n^2}$. At a constant factor increase in T, this failure probability decreases *exponentially*.

We have now bounded the probability that a single machine fails to uniquely access the shared resource in T time steps, but our goal was to serve all n processes. Fortunately, this is a ripe opportunity for the union bound (8). For all $i \in [n]$, let $\mathcal{E}_i$ denote the event that after T time steps, the i^{th} process has not been successfully served. We have argued that for $T > 8n \log(n)$, $\Pr[\mathcal{E}_i] \leq \frac{1}{n^2}$, so an application of (8) gives:

$$\Pr\left[\bigcup_{i \in [n]} \mathcal{E}_i\right] \leq \sum_{i \in [n]} \frac{1}{n^2} = \frac{1}{n}.$$

The left-hand side above is the probability that any one process has not been successfully served. Therefore, by taking complements, all processes are served with success probability at least $1 - \frac{1}{n}$, which is a very strong bound as n grows large. It is surprising that we are able to achieve such

a bound in only $O(n\log(n))$ time steps using a simple contention resolution algorithm with no inter-process communication, given that n time steps are necessary even with communication.

This application is a basic example of the power of independent repetition in randomized algorithm design. Even though each individual time step is extremely unlikely to serve any given machine (with a success probability $\leq \frac{1}{n}$), if we repeat trials enough times (e.g., $\approx n\log(n)$ trials), even the success probability for *all processes* becomes very close to 1.

Our algorithm had the interesting property of having a much smaller failure probability at a slight runtime increase: for instance, taking $T > 16n\log(n)$ improves the failure probability to $n \cdot \frac{1}{n^4} = \frac{1}{n^3}$. Randomized algorithms with failure probability inverse-polynomial in the input size n are often colloquially said to succeed "with high probability." This is often the gold standard in randomized algorithm design, compared to an algorithm with, e.g., a $\frac{1}{10}$ failure probability, for the same reason that we typically use asymptotic runtimes to benchmark algorithm performance: all else held equal, we should prefer the algorithm that scales better to large inputs.

3 Concentration

In Section 2.2, we saw a basic example of boosting success probabilities via repetition. Specifically, our main observation was that if we run an algorithm $\mathcal{A}$ and some desired outcome $\mathcal{E}$ occurs with probability $\geq p$, then if we run $\mathcal{A}$ for $\approx \frac{1}{p}$ times, it is reasonably likely that we will see at least one successful run. To extrapolate slightly, if we let $T > \frac{1}{p}\log(\frac{1}{\delta})$, then by a variant of (9),

$$\Pr\left[\mathcal{E} \text{ does not occur in } T \text{ runs of } \mathcal{A}\right] = (1-p)^T \leq \left((1-p)^{\frac{1}{p}}\right)^{\log(\frac{1}{\delta})} \leq \left(\frac{1}{e}\right)^{\log(\frac{1}{\delta})} \leq \delta. \tag{10}$$

Such a strategy is effective when we can efficiently verify whether a run of $\mathcal{A}$ succeeded, e.g., in our FindPivot example in Section 1, verification requires $O(n)$ time. However, note that in this example, the cost of running $\mathcal{A}$ was only $O(1)$, whereas verification was much more expensive. This begs the question: can we directly boost success probability with another strategy?

In this section, we take a broader viewpoint on this question, and introduce *concentration* as a tool for understanding the behavior of random variables. We then show how to use concentration with simple aggregation tricks like taking the mean or median to boost success probabilities.

3.1 Concentration from variance

The basic statistic we use to measure concentration in this section is the *variance*:

$$\mathrm{Var}[X] := \mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2. \tag{11}$$

The variance is always nonnegative: to remember this, the function $f(x) = x^2$ is convex, so $f(\mathbb{E}[X])$ stays "below the line" predicted by $\mathbb{E}[f(X)]$. The reason why it is useful is due to the identity

$$\begin{aligned}\mathbb{E}\left[(X - \mathbb{E}[X])^2\right] &= \mathbb{E}\left[X^2\right] - 2\mathbb{E}\left[X \cdot \mathbb{E}[X]\right] + \mathbb{E}[X]^2 \\ &= \mathbb{E}\left[X^2\right] - 2\mathbb{E}\left[X\right]^2 + \mathbb{E}\left[X\right]^2 = \mathrm{Var}\left[X\right].\end{aligned} \tag{12}$$

The formula (12) is another common definition of the variance, and shows that $\sqrt{\mathrm{Var}[X]}$, the *standard deviation*, is a decent proxy for how large we expect the "spread" $|X - \mathbb{E}[X]|$ to be on average. We will give tools to make this intuition precise in this section.

Before we do, however, we develop some helpful calculation rules for the variance. Two commonly-used properties of the expectation are that if X is a random variable, $\mathbb{E}[cX] = c\mathbb{E}[X]$ for any constant multiple $c \in \mathbb{R}$, and if Y is another random variable, $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$ (linearity of expectation). An analog of the first statement is also true for the variance, by (12):

$$\mathrm{Var}\left[cX\right] = \mathbb{E}\left[(cX)^2\right] - \mathbb{E}\left[cX\right]^2 = c^2\left(\mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2\right) = c^2\mathrm{Var}[X]. \tag{13}$$

This makes sense intuitively, as we would expect cX to have an "average spread" about c times larger than X, and the variance captures the average squared spread. However, the analog of the

linearity of expectation is not true in general, and may not hold for dependent X, Y:

$$\begin{aligned}\operatorname{Var}[X+Y] &= \mathbb{E}\left[(X+Y)^2\right] - \mathbb{E}[X+Y]^2 \\ &= \mathbb{E}\left[X^2\right] + 2\mathbb{E}[XY] + \mathbb{E}\left[Y^2\right] - \mathbb{E}\left[X^2\right] - 2\mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}\left[Y^2\right] \\ &= \operatorname{Var}[X] + \operatorname{Var}[Y] + 2\left(\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]\right) \\ &= \operatorname{Var}[X] + \operatorname{Var}[Y], \text{ if } X, Y \text{ are independent.}\end{aligned} \tag{14}$$

In the last line, we used that independence of X, Y implies $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ (see discussion in Section 6, Part I). As a word of warning, if this does not hold, then we cannot use (14).

With these calculation rules in mind, we introduce our main tools for using the variance to reason about concentration. We begin with *Markov's inequality* as a warmup.

Lemma 1 (Markov's inequality)**.** *For any nonnegative random variable X and any $\delta \in (0,1)$,*

$$\Pr\left[X \geq \frac{\mathbb{E}[X]}{\delta}\right] \leq \delta.$$

Proof. Suppose otherwise, i.e., that $X \geq \frac{\mathbb{E}[X]}{\delta}$ with probability $> \delta$. Then, we have a contradiction:

$$\begin{aligned}\mathbb{E}[X] &= \Pr\left[X \geq \frac{\mathbb{E}[X]}{\delta}\right] \cdot \mathbb{E}\left[X \mid X \geq \frac{\mathbb{E}[X]}{\delta}\right] + \Pr\left[X < \frac{\mathbb{E}[X]}{\delta}\right] \cdot \mathbb{E}\left[X \mid X < \frac{\mathbb{E}[X]}{\delta}\right] \\ &\geq \Pr\left[X \geq \frac{\mathbb{E}[X]}{\delta}\right] \cdot \mathbb{E}\left[X \mid X \geq \frac{\mathbb{E}[X]}{\delta}\right] > \delta \cdot \frac{\mathbb{E}[X]}{\delta} = \mathbb{E}[X].\end{aligned}$$

□

For slightly more intuition on the proof of Lemma 1, consider $\delta = \frac{1}{2}$; it is just saying that a random variable cannot be more than twice its expectation, more than half the time, or this contribution would already overshoot its expectation. Lemma 1 generalizes this to arbitrary δ.

While Lemma 1 gives us a basic sense of how a random variable deviates from its mean, it is unsatisfactory in a few ways. First, it only applies to nonnegative X, and second, it can only prove loose bounds of the form, $X \leq 10\mathbb{E}[X]$ except with probability $\frac{1}{10}$. Fortunately, the following extension of Lemma 1 is usually more applicable and lets us prove much stronger statements.

Lemma 2 (Chebyshev's inequality)**.** *For any random variable X and any $\delta \in (0,1)$,*

$$\Pr\left[|X - \mathbb{E}[X]| \geq \frac{\sqrt{\operatorname{Var}[X]}}{\sqrt{\delta}}\right] \leq \delta.$$

Proof. Applying Lemma 1 with $X \leftarrow (X - \mathbb{E}[X])^2$, we have from (12) that

$$\Pr\left[(X - \mathbb{E}[X])^2 \geq \frac{\operatorname{Var}[X]}{\delta}\right] \leq \delta.$$

The claim follows because

$$(X - \mathbb{E}[X])^2 \geq \frac{\operatorname{Var}[X]}{\delta} \iff |X - \mathbb{E}[X]| \geq \frac{\sqrt{\operatorname{Var}[X]}}{\sqrt{\delta}}.$$

□

The precise dependence of Chebyshev's inequality on δ is not extremely important towards its qualitative message, which is that, with a large constant probability, any random variable X lies within a few standard deviations of its mean. For example, taking $\delta = \frac{1}{4}$ in Lemma 2 gives

$$\Pr\left[X \in \left[\mathbb{E}[X] + 2\sqrt{\operatorname{Var}[X]}, \mathbb{E}[X] - 2\sqrt{\operatorname{Var}[X]}\right]\right] \geq \frac{3}{4}. \tag{15}$$

3.2 Boosting concentration

While Lemma 2 gives us a preliminary interval where a random variable will likely lie, it loses somewhat large constant factors in the bound, and its range depends polynomially on the failure probability δ. We would prefer that our algorithm's guarantees depend on $\log(\frac{1}{\delta})$, as was the case in (10). In this section, we give a simple framework for improving both of these dependences.

Mean boosting. The first technique is a simple strategy for improving the "radius" of the interval predicted by Lemma 2 by a constant factor. In particular, suppose we have the ability to generate multiple independent copies of X. This could be the case if X is the output of an algorithm $\mathcal{A}$ is trying to estimate some quantity; we just call $\mathcal{A}$ independently in parallel multiple times.

The key observation is that if $\{X_i\}_{i\in[k]}$ are independent copies of X, then if we let $\bar{X} := \frac{1}{k}\sum_{i\in[k]} X_i$ denote their average, we have from (13) and (14) that

$$\text{Var}\left[\bar{X}\right] = \frac{1}{k^2}\text{Var}\left[\sum_{i\in[k]} X_i\right] = \frac{1}{k^2}\sum_{i\in[k]}\text{Var}[X_i] = \frac{1}{k}\text{Var}[X]. \tag{16}$$

Here, independence was crucially used in the second equality (i.e., in (14)).

The takeaway from (16) is that if we can run an experiment multiple times, we can decrease the variance (and hence the length of the predicted interval due to Lemma 2) by a constant factor. More generally, if we want to decrease said interval length by a factor of $\epsilon \in (0,1)$, we should take $k \approx \epsilon^{-2}$. As an example application, if $\bar{X}$ is the average of ϵ^{-2} independent random copies of X, then in place of (15), we have the much tighter bound

$$\Pr\left[\bar{X} \in \left[\mathbb{E}[X] - 2\epsilon\sqrt{\text{Var}[X]}, \mathbb{E}[X] + 2\epsilon\sqrt{\text{Var}[X]}\right]\right] \geq \frac{3}{4}.$$

Median boosting. The second technique is a generic tool for improving the failure probability of an interval prediction to $\delta \in (0,1)$, at an $O(\log(\frac{1}{\delta}))$ overhead in runtime.

Lemma 3. *Let $I \subset \mathbb{R}$ be an interval, and suppose that for some random variable X, $\Pr[X \in I] \geq \frac{3}{4}$. Then if $\{X_i\}_{i\in[k]}$ are independent copies of X for $k \geq 12\log(\frac{1}{\delta})$, and $\widehat{X}$ is the median of $\{X_i\}_{i\in[k]}$,*

$$\Pr\left[\widehat{X} \in I\right] \geq 1 - \delta.$$

Proof. We first claim that if $> \frac{k}{2}$ of the independent copies fall in I, then so will the median $\widehat{X}$. To see this, if $\widehat{X}$ is right of I, then so are at least half of the copies, a contradiction; a similar argument rules out $\widehat{X}$ being left of I. Thus our goal is to argue that except with probability δ, at least half of the copies fall in I. Each copy falls in I with some probability $p \geq \frac{3}{4}$.

The claim follows from a standard concentration inequality known as a *Chernoff bound* (see e.g., [Wik24]). This goes beyond the scope of this class, so we will not prove it here, but we do wish to give some intuition on why it applies in this case. Observe that we are trying to bound

$$\sum_{i\geq\frac{k}{2}}\binom{k}{i}(1-p)^i p^{k-i} \leq \sum_{i\geq\frac{k}{2}}\binom{k}{i}\left(\frac{1}{4}\right)^i\left(\frac{3}{4}\right)^{k-i} \leq \delta,$$

where the i^{th} term captures the probability that exactly i of the copies fall outside of I. The main idea is that this sequence decreases geometrically: $\binom{k}{i}$ is decreasing for $i \geq \frac{k}{2}$, and $(\frac{1}{4})^i(\frac{3}{4})^{k-i}$ decreases by a factor of 3 as i increases by 1. A geometric sequence is dominated by its largest term, which is the first term in this case, so it suffices to show

$$\binom{k}{k/2}\left(\frac{1}{4}\right)^{\frac{k}{2}}\left(\frac{3}{4}\right)^{\frac{k}{2}} \leq 2^k\left(\frac{3}{16}\right)^{\frac{k}{2}} \leq \left(\frac{3}{4}\right)^{\frac{k}{2}} = O(\delta).$$

In the first inequality, we used $\binom{k}{k/2} \leq \sum_{i=0}^{k}\binom{k}{i} = 2^k$. The claim follows if $k = \Theta(\log(\frac{1}{\delta}))$. □

Combining median boosting with mean boosting, we have shown that if Chebyshev's inequality (Lemma 2) predicts that $X \in I := [\mathbb{E}[X] - R, \mathbb{E}[X] + R]$ with probability $\geq \frac{3}{4}$, then by sampling $O(\log(\frac{1}{\delta}) \cdot \epsilon^{-2})$ independent copies of X, we can produce an estimate $\widehat{X}$ such that the much stronger guarantee $\widehat{X} \in [\mathbb{E}[X] - \epsilon R, \mathbb{E}[X] + \epsilon R]$ holds with probability $\geq 1 - \delta$.

Notably, this works regardless of our ability to ascertain whether a given copy of X fell in I. Thus, for problems where our goal is to estimate an unknown quantity $\mathbb{E}[X]$, the strategies in this section can generically be used to boost performance of a simple estimator X with bounded variance.

3.3 Morris counter

We now give a famous application of the machinery in Section 3.2. Just as contention resolution (Section 2.2) demonstrated the power of randomness in saving on communication as a computational resource, this example will show how it can be used to save on space.

The problem is as follows: we want to build a data structure that initializes $i \to 0$ and supports two operations, Count() and Report(). Each time Count() is called, i is incremented, and finally after n calls to Count(), we call Report(). The goal of Report() is to report an estimate of n. This is trivial if we are allowed $O(\log(n))$ space (we can just write the binary representation of n), so the interesting problem is to design a data structure using less space.

This is motivated by settings requiring many counters of many large numbers. For example, this could easily be the case if you are running a web crawler or search engine: even a constant factor savings in the number of bits required per counter could make a huge impact on your overall space usage. Indeed, the original motivation of Morris [Mor78], who gave the first nontrivial algorithm for this problem, was to count $26^3 \geq 10^4$ different trigrams for implementing a spellchecker.

We consider the following *Morris counter* algorithm, which stores a random variable X initialized at $X \leftarrow 0$. It implements Count() as follows: with probability 2^{-X}, $X \leftarrow X+1$. Its implementation of Report() is to return $2^X - 1$. Intuitively, the goal of X is to approximate $\log(i+1)$, which it simulates by rarely incrementing. Note that as long as $X = O(\log(n))$, we can store it with a much smaller $O(\log\log(n))$ bits, compared to directly storing i. We prove the following claims.

Lemma 4. *For all $0 \leq k \leq n$, let X_k denote the value of the random variable X after* Count() *is called exactly k times. Then for all $0 \leq k \leq n$,*

$$\mathbb{E}\left[2^{X_k}\right] = k + 1.$$

Proof. We use induction. This is clear when $k = 0$, so it remains to show $\mathbb{E}[2^{X_{k+1}}] = \mathbb{E}[2^{X_k}] + 1$ for all $k \geq 0$, i.e., that our estimator's expectation grows by 1 when Count() is called. To see this,

$$\begin{aligned}
\mathbb{E}\left[2^{X_{k+1}}\right] &= \sum_{j \geq 0} \Pr[X_k = j] \cdot \left(\left(1 - \frac{1}{2^j}\right) \cdot 2^j + \frac{1}{2^j} \cdot 2^{j+1}\right) \\
&= \sum_{j \geq 0} \Pr\left[X_k = j\right] \cdot \left(2^j + 1\right) \\
&= \sum_{j \geq 0} \Pr[X_k = j] \cdot 2^j + \sum_{j \geq 0} \Pr[X_k = 1] = \mathbb{E}\left[2^{X_k}\right] + 1.
\end{aligned} \tag{17}$$

□

Lemma 5. *In the setting of Lemma 5, for all $1 \leq k \leq n$,*

$$\mathrm{Var}\left[2^{X_k}\right] \leq 4k^2.$$

Proof. Recall that $\mathrm{Var}[2^{X_k}] = \mathbb{E}[4^{X_k}] - \mathbb{E}[2^{X_k}]^2 \leq \mathbb{E}[4^{X_k}]$. We claim $\mathbb{E}[4^{X_k}] = \frac{3}{2}k^2 + \frac{3}{2}k + 1$, which

is clear when $k = 0$. The general claim follows analogously to (17): by induction,

$$\begin{aligned}\mathbb{E}\left[4^{X_{k+1}}\right] &= \sum_{j\geq 0} \Pr\left[X_k = j\right] \cdot \left(\left(1 - \frac{1}{2^j}\right) \cdot 4^j + \frac{1}{2^j} \cdot 4^{j+1}\right) \\ &= \sum_{j\geq 0} \Pr\left[X_k = j\right] \cdot 4^j + 3 \sum_{j\geq 0} \Pr\left[X_k = j\right] \cdot 2^j \\ &= \mathbb{E}\left[4^{X_k}\right] + 3\mathbb{E}\left[2^{X_k}\right] = \left(\frac{3}{2}k^2 + \frac{3}{2}k + 1\right) + 3\left(k+1\right) = \frac{3}{2}\left(k+1\right)^2 + \frac{3}{2}\left(k+1\right) + 1.\end{aligned}$$

□

By applying Lemmas 4 and 5 with $k \leftarrow n$, we obtain an estimator $Y \leftarrow 2^X - 1$ that satisfies $\mathbb{E}[Y] = n$ and $\mathrm{Var}[Y] \leq 4n^2$. Thus, directly applying (15) gives an essentially useless reporting range of $Y \in [n - 4n, n + 4n]$. However, by using the boosting techniques in Section 3.2, we can considerably improve this estimate: we can maintain an average of $O(\frac{1}{\epsilon^2})$ copies of the algorithm to form a better estimator $\bar{Y}$, and then take the median $\widehat{Y}$ of $O(\log(\frac{1}{\delta}))$ copies of $\bar{Y}$, to achieve

$$\widehat{Y} \in [(1-\epsilon)n, (1+\epsilon)n] \text{ with probability } \geq 1 - \delta. \tag{18}$$

For moderate ϵ and δ, this yields an algorithm with space complexity $O(\log\log(n))$, an *exponential* improvement to the naïve algorithm. By being more careful with the implementation of the Morris counter, e.g., incrementing with probability $(1+\alpha)^{-X}$ instead of 2^{-X} for some $\alpha \approx \epsilon^2\delta$ and early terminating appropriately, [NY22] show that we can achieve the guarantee (18) using only $O(\log\log(\frac{n}{\delta}) + \log(\frac{1}{\epsilon}))$ space, which for many interesting regimes is $o(\log(n))$.

4 Sampling

In this section, we introduce *sampling* as an algorithmic task. Sampling, i.e., producing samples $\omega \in \Omega$ from some distribution over Ω that is either explicitly or implicitly given, is an inherently randomized problem. It is one of the cornerstone tools in modern machine learning and data science, representing many facets of the algorithmic toolkit. For example, it is used for *parameter estimation* in statistics, and *counting* or *integration* in numerical analysis.

More recently, sampling has unlocked new algorithmic capabilities, e.g., *uncertainty quantification* and *generation*. To explain the former, in settings where we are uncertain in a learning task, it is often better to represent our knowledge in the form of a distribution. We can estimate quantities like the variance of the distribution from samples, which helps us quantify our uncertainty. For example, such techniques can be used to *calibrate* large language models so they can accurately reflect their knowledge. Regarding the latter, a common task in generative modeling is to take samples from a distribution π of interest, e.g., all pictures of cats or code snippets implementing MergeSort. Given these samples, generative models attempt to produce new samples from π.

This section provides a basic overview of some simple, but very influential, ideas in sampling algorithms. We begin with a classical sampling problem in Section 4.1, as an example of a more general technique, *rejection sampling*, developed in Section 4.2. In Section 4.3, we conclude by describing one application of rejection sampling to bias correction in Markov chain Monte Carlo (MCMC) methods, the predominant sampling framework of today.

4.1 Reservoir sampling

In this section we consider the following problem: we are given a stream of items $\omega_1, \omega_2, \ldots, \omega_n \in \Omega$, and our goal is to return a uniformly random item from the stream, i.e., ω_i where $i \in [n]$ is chosen uniformly at random. The only catch is that we do not know n in advance, so the stream could stop at any time. Another way of viewing this problem (more in line with Section 3.3) is that we are building a data structure that supports two operations: $\mathsf{Stream}(\omega)$, which inputs ω into the stream, and $\mathsf{Report}()$, which outputs a uniformly random item amongst those in the stream.

A naïve way to solve this problem is to simply store all streamed items, and then when $\mathsf{Report}()$ is called after n calls to Stream, we randomly sample $i \in [n]$ and output the i^{th} item. However, this

implementation takes $O(n)$ space, which can be very expensive when the stream itself is long (e.g., unique visitors to a popular website). The solution in this section, called the *reservoir sampling* algorithm, solves the given data structure problem using only $O(1)$ words of space.[2]

Our data structure maintains a counter i, initialized to 0, that represents the number of times Stream has been called. It also maintains a "current" item, $\bar{\omega}$, initialized to **None**, which we can think of as being stored in a reservoir. On an iteration where Stream(ω) is called, we update

$$\bar{\omega} \leftarrow \begin{cases} \bar{\omega} & \text{with probability } \frac{i}{i+1} \\ \omega & \text{with probability } \frac{1}{i+1} \end{cases}. \tag{19}$$

We then update $i \leftarrow i+1$. In other words, we evict the current reservoir item with probability $\frac{1}{i+1}$, replacing it with the new streamed item. When Report() is called, we simply return $\bar{\omega}$.

It is clear that reservoir sampling only uses $O(1)$ space, as it only needs to store a counter and a single reservoir item. The key correctness claim is that for any $n \in \mathbb{N}$, after Stream is called n times with items $\{\omega_i\}_{i\in[n]}$, the resulting distribution on $\bar{\omega}$ is uniform over the n items. We give two proofs of this fact, the second of which suggests a more general design technique.

Direct calculation. Suppose that Stream has been called n times with items $\{\omega_i\}_{i\in[n]}$, and let $\pi : [n] \to [0,1]$ be the resulting distribution of the reservoir item $\bar{\omega}$, i.e., $\pi(i) := \Pr[\bar{\omega} = \omega_i]$. Our goal is to show that $\pi(i) = \frac{1}{n}$ for all $i \in [n]$. To do this, we can simply calculate:

$$\pi(i) = \frac{1}{i} \cdot \frac{i}{i+1} \cdot \frac{i+1}{i+2} \cdot \ldots \cdot \frac{n-1}{n} = \frac{1}{i} \cdot \frac{i}{n} = \frac{1}{n}. \tag{20}$$

Here, the first equality used that the only way that $\bar{\omega} = \omega_i$ is if ω_i evicted the reservoir in the i^{th} call to Stream, and then it was not evicted in any of the future calls, $i+1 \le j \le n$. By using the formula (19), this probability is $\frac{1}{i}$ for eviction in the i^{th} call (as Stream had only been called $i-1$ times previously when ω_i is streamed), and $\frac{j-1}{j}$ for non-eviction in all future calls $i+1 \le j \le n$. The second equality telescoped terms between consecutive non-eviction calls.

Fixing the distribution. The proof in (20) is somewhat mysterious, and seemed very specialized to our setting. Here we give a different interpretation of our proof that is suggestive of a broader technique. Suppose that items $\{\omega_i\}_{i\in[n]}$ have been streamed at the time Report is called, and for all $j \in [n]$, let π_j be the distribution of the index of the reservoir item $\bar{\omega}$ after the first j calls to Stream. Our goal is to show π_n is the uniform distribution on $[n]$. We instead prove the stronger claim that for all $j \in [n]$, π_j is the uniform distribution on $[j]$, by induction.

After the first call to Stream, the update (19) shows that $\bar{\omega} = \omega_1$ with probability 1 as claimed. Now inductively suppose that π_j is uniform over $[j]$, for some $j \in [n-1]$. We can directly compute, using the formula (19), that for all $k \in [j+1]$,

$$\pi_{j+1}(k) = \begin{cases} \frac{j}{j+1}\pi_j(k) & k \in [j] \\ \frac{1}{j+1} & k = j+1 \end{cases}.$$

Here the first case considers the chance that ω_k is the current reservoir item after j calls completed. The proof concludes using the inductive hypothesis:

$$\frac{j}{j+1}\pi_j(k) = \frac{j}{j+1} \cdot \frac{1}{j} = \frac{1}{j+1}.$$

Intuitively, this proof "fixes" the distribution π_j to make it match π_{j+1}, by adjusting the weights it gives to $[j]$ to match the desired output, and then inserting a weight of $\frac{1}{j+1}$ on the $(j+1)^{\text{th}}$ item. This idea of correcting a known distribution is the key idea of rejection sampling.

4.2 Rejection sampling

Suppose you have a distribution π that you wish to sample from, but it is difficult to do so directly. However, suppose further that you have the ability to sample from a "nearby" distribution $\mu \neq \pi$, which may be simpler than the actual target π, but which also approximates π well. The idea of

[2] As usual, we work in the word RAM model (Section 7, Part I), where each item can be stored in $O(1)$ space.

rejection sampling is to "fix" a sample from μ based on its relative probabilities according to π and μ, so that the output distribution is instead π, and the overall sampler is efficient.

The general idea of basing sampling complicated distributions on simpler distributions is a powerful idea in sampler design in general, and examples abound: for instance, the *Box-Muller transform* shows how to generate random variables from π, the standard normal distribution, given the ability to sample random variables from μ, the uniform distribution on $[0, 1]$. Often these transform-based tools require fairly explicit manipulation of the formulas for π and μ. Here we outline a more flexible technique that requires only very weak knowledge of the forms of π and μ.

In more detail, suppose π and μ are both distributions over a sample space Ω, and let Ω be discrete for simplicity (although the same technique generalizes essentially verbatim to the continuous setting). Further, suppose we know that for some nonnegative $P : \Omega \to \mathbb{R}_{\geq 0}$ and $Q : \Omega \to \mathbb{R}_{\geq 0}$,

$$\begin{aligned} \pi(\omega) = \frac{P(\omega)}{N_P},\ \mu(\omega) = \frac{Q(\omega)}{N_Q},\ \text{for all } \omega \in \Omega, \\ \text{where } N_P := \sum_{\omega \in \Omega} P(\omega),\ N_Q := \sum_{\omega \in \Omega} Q(\omega). \end{aligned} \tag{21}$$

Intuitively, P and Q are "unnormalized" versions of π and μ, and N_P, N_Q are the *normalization constants* that allow π and μ to be true probability distributions, i.e., so that $\sum_{\omega \in \Omega} \pi(\omega) = \sum_{\omega \in \Omega} \mu(\omega) = 1$. In this case, we use the notation $\pi \propto P$ to mean that π is the corresponding normalized distribution. For example, if $\Omega = [4]$, $P = \{1, 2, 1, 0\}$, and $Q = \{1, 1, 1, 1\}$, then $N_P = N_Q = 4$, $\pi \propto P$ is the distribution on $[4]$ that puts weight $\frac{1}{4}$ on items in $\{1, 3\}$ and weight $\frac{1}{2}$ on item 2, and $\mu \propto Q$ is the uniform distribution on $[4]$ putting weight $\frac{1}{4}$ on every item.

Unnormalized distributions give us an extra degree of flexibility to work with, that allows us to "bump up" a distribution μ pointwise to an unnormalized variant Q, at the cost of increasing N_Q. This flexibility will be used in our rejection sampling framework shortly. In practice, it is often convenient to specify "weights" of items (e.g., counts in a dataset), that we wish to sample proportionally to. Additionally, observe that while unnormalizing a distribution changes the weight assigned to each item $\omega \in \Omega$, it preserves *ratios* between items: following (21), for any $\omega, \omega' \in \Omega$,

$$\frac{P(\omega)}{P(\omega')} = \frac{N_P \cdot \pi(\omega)}{N_P \cdot \pi(\omega')} = \frac{\pi(\omega)}{\pi(\omega')}.$$

The intuition is that unnormalized distributions reflect the relative importance of items.

We now have assembled all of the technical tools we need to formalize the rejection sampling framework (the main other helpful background, *conditional distributions*, is reviewed thoroughly in Section 6.2, Part I). We state the framework in Algorithm 1.

Algorithm 1: RejectionSample(μ, P, Q)

Input: Sample access to a distribution μ over a finite sample space Ω, $P : \Omega \to \mathbb{R}_{\geq 0}$, $Q : \Omega \to \mathbb{R}_{\geq 0}$ such that μ, Q have the relationship in (21), and such that

$$P(\omega) \leq Q(\omega) \text{ for all } \omega \in \Omega. \tag{22}$$

while True do
- $\omega \leftarrow$ sample from μ
- $p \leftarrow \frac{P(\omega)}{Q(\omega)}$
- $Z \leftarrow 1$ with probability p, otherwise $Z \leftarrow 0$
- **if** $Z == 1$ **then**
  - **return** ω
- **end**

end

The following analysis of Algorithm 1 is the main claim of the section.

Lemma 6. *Let Ω be a finite sample space, let π and μ be distributions over Ω, and suppose P and Q are unnormalized distributions satisfying (21). Further, suppose (22) holds. Algorithm 1 with inputs μ, P, Q samples exactly from π, and uses $\frac{N_Q}{N_P}$ samples from μ in expectation.*

Proof. Algorithm 1 repeatedly runs loops of Lines 1 through 8, until the random variable Z evaluates to 1, at which point it terminates. We want to understand two things: the probability that $Z = 1$ on any given run, and the conditional distribution of ω on $Z = 1$.

We begin by discussing the former. If $\omega \in \Omega$ is sampled, it is returned with probability $\frac{P(\omega)}{Q(\omega)}$. Therefore, the overall acceptance probability is

$$\sum_{\omega \in \Omega} \mu(\omega) \cdot \frac{P(\omega)}{Q(\omega)} = \sum_{\omega \in \Omega} \frac{Q(\omega)}{N_Q} \cdot \frac{P(\omega)}{Q(\omega)} = \frac{\sum_{\omega \in \Omega} P(\omega)}{N_Q} = \frac{N_P}{N_Q}.$$

Each loop run terminates with this probability, which we model as a coin that comes up heads with probability $\frac{N_P}{N_Q}$. The expected number of tosses before our first heads follows from (7).

Now, consider the conditional distribution on $Z = 1$. It puts weight on $\omega \in \Omega$ proportional to

$$Q(\omega) \cdot \frac{P(\omega)}{Q(\omega)} = P(\omega),$$

where the first term is from the original sample $\omega \sim \mu$, and the second is the acceptance probability. We know that the corresponding normalized distribution is π as claimed. □

Lemma 6 makes our goal in designing an instantiation of Algorithm 1 clear: we want to pick two unnormalized distributions P and Q, such that $\pi \propto P$ for our target distribution π, and $\mu \propto Q$ for some "simple" distribution μ that we can sample from. Moreover, for Algorithm 1 to terminate quickly, the shapes of P and Q should approximate each other in the following sense.

1. $0 \leq P(\omega) \leq Q(\omega)$ for all $\omega \in \Omega$, so pictorially, P stays at or below Q pointwise.
2. $\frac{\sum_{\omega \in \Omega} Q(\omega)}{\sum_{\omega \in \Omega} P(\omega)}$ is small, so the algorithm terminates quickly. Intuitively, this is saying that Q is not too much of an overestimate of P on average over the universe Ω.

As an illustrative example in continuous probability,[3] suppose our goal is to output a uniformly random point from the unit-radius circle in $\mathbb{R}^2$, i.e., the set $\mathcal{C} := \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$.

One way to do this is to let π be the desired uniform distribution over the circle $\mathcal{C}$, and $P(\omega) = 1$ for all $\omega \in \mathcal{C}$. We will also let $\mathcal{S}$ denote the unit square $[-1, 1] \times [-1, 1]$, μ be the uniform distribution over $\mathcal{S}$, and $Q(\omega) = 1$ for all $\omega \in \mathcal{S}$. Clearly, $0 \leq P(\omega) \leq Q(\omega)$ for all $\omega \in \mathcal{S}$, and moreover by the rules of continuous probability (see Section 6.1, Part I), N_P and N_Q are just the respective areas of $\mathcal{C}$ and $\mathcal{S}$ in this example. Also, it is simple to generate a sample from $\mu \propto Q$, by sampling two independent uniform random numbers in $[-1, 1]$. Thus, the expected number of iterations is

$$\frac{4}{\pi} \approx 1.273\ldots$$

and Algorithm 1 is very likely to terminate using just a few samples, due to (10).

4.3 Metropolis-Hastings MCMC

We now briefly describe a powerful application of Section 4.2: the Metropolis-Hastings implementation of the Markov chain Monte Carlo (MCMC) method. This algorithm, like the FFT and simplex method, was named a top-10 most influential algorithm of the 20$^{\text{th}}$ century [DS00].

Suppose that we are trying to sample from some distribution π over a universe Ω, which could be discrete or even continuous. The key idea in MCMC is to design "local distributions" $\mathcal{T}_\omega : \Omega \to [0, 1]$ for each $\omega \in \Omega$, which can be viewed as specifying "transition probabilities" for a random walk over Ω. Ideally, it should be possible to efficiently sample from $\mathcal{T}_\omega$ to implement the random walk. Moreover, these local distributions should have the property that if ω is already distributed according to π, then after one step of the corresponding random walk using the $\{\mathcal{T}_\omega\}_{\omega \in \Omega}$ to govern the next choice of ω, it should still be the case that $\omega \sim \pi$.

Intuitively, the strategy in MCMC is to simulate the random walk for many iterations until it "mixes," and ω becomes distributed according to a "stationary distribution" set to π. Under

[3] Thanks to Arun Jambulapati for this suggestion.

relatively mild conditions, the stationary distribution of a MCMC algorithm is unique. However, it can be rather difficult to find these local distributions $\mathcal{T}_\omega$ that magically happen to align into a desired global distribution π over Ω, especially when π itself is not explicit (e.g., is only given as proportional to weights, and depends on unknown normalization constants).

The Metropolis-Hastings algorithm uses rejection sampling to design $\mathcal{T}_\omega$. The idea is: let $\mathcal{P}_\omega$ be some simple distribution that coarsely resembles π zoomed in on a small neighborhood of ω, e.g., a uniform distribution, or a small Gaussian. We will use rejection sampling to "fix" $\mathcal{P}_\omega$ in a way that makes π the overall stationary distribution. Concretely (in the discrete case), we let

$$\mathcal{T}_\omega(\omega') = \mathcal{P}_\omega(\omega') \cdot \min\left(1, \frac{\pi(\omega')\mathcal{P}_{\omega'}(\omega)}{\pi(\omega)\mathcal{P}_\omega(\omega')}\right). \tag{23}$$

This has a simple interpretation as a rejection sampling scheme, where we first draw $\omega' \sim \mathcal{P}_\omega$, and then accept it with some probability in $[0, 1]$. This acceptance probability can actually be evaluated even if both π and $\mathcal{P}_\omega$ are only given up to normalization constants (i.e., as unnormalized distributions), which turns out to be important in many applications. Our goal, as in all rejection sampling schemes, is to make sure that the rejection probability in (23) is not too small on average, which can be interpreted as $\mathcal{P}_\omega$ capturing the "local shape" of π around ω.

Let us briefly argue that the $\mathcal{T}_\omega$ we gave actually do have π as a stationary distribution. Let $\omega \sim \pi$, and let $\omega' \sim \mathcal{T}_\omega$ be the next state of the random walk. We compute for any $\bar{\omega} \in \Omega$:

$$\begin{aligned}
\Pr[\omega' = \bar{\omega}] &= \sum_{\omega \in \Omega} \pi(\omega)\mathcal{T}_\omega(\bar{\omega}) \\
&= \sum_{\omega \in \Omega} \min\left(\pi(\omega)\mathcal{P}_\omega(\bar{\omega}), \pi(\bar{\omega})\mathcal{P}_{\bar{\omega}}(\omega)\right) \\
&= \sum_{\omega \in \Omega} \pi(\bar{\omega})\mathcal{T}_{\bar{\omega}}(\omega) = \pi(\bar{\omega}).
\end{aligned}$$

Thus, ω' is also distributed $\sim \pi$ after the Metropolis-Hastings correction, as promised.

To give a very simple application, let π be the uniform distribution over some set $\mathcal{K} \subset \mathbb{R}^d$, and let each $\mathcal{P}_\omega$ be the uniform distribution over a small ball centered at $\omega \in \mathcal{K}$. If we apply the Metropolis-Hastings correction (23), then because $\omega \in \mathcal{K}$ is always preserved, the acceptance probability is simply 1 if $\omega' \in \mathcal{K}$ and 0 otherwise. Thus, the resulting Markov chain (called the *ball walk*) just repeatedly draws ω' in a small ball around ω until it finds a point in $\mathcal{K}$, and then moves there. Its rate of convergence ends up being proportional to two things: the size of the balls governing the $\{\mathcal{P}_\omega\}_{\omega \in \mathcal{K}}$, and the amount of "surface area" of $\mathcal{K}$ at a certain coarseness, which governs the proability that a transition steps outside the set and is rejected.

In general, the Metropolis-Hastings framework for MCMC lets us be quite creative with designing the local distributions $\mathcal{P}_\omega$, as long as we have ways to manipulate their density functions to implement the correction step (23). This idea is widely-used in many samplers today.

Further reading

For more on Section 2, see Chapters 14 to 16, [LLM10] and Chapters 13.1 and 13.3, [KT05].

For more on Section 3, see Chapters 18 to 19, [LLM10] and Chapter 13.9, [KT05].

For more on Section 4, see Chapter 20, [LLM10].

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of Mathematics*, 160(2):781–793, 2004.

[DS00] J. Dongarra and F. Sullivan. Guest Editors Introduction to the top 10 algorithms . *Computing in Science & Engineering*, 2(01):22–23, 2000.

[KT05] Jon Kleinberg and Éva Tardos. *Algorithm Design*. 2005.

[LLM10] Eric Lehman, F.T̃homson Leighton, and Albert R. Meyer. *Mathematics for Computer Science*. 2010.

[Mor78] Robert H. Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[NY22] Jelani Nelson and Huacheng Yu. Optimal bounds for approximate counting. In *PODS '22: International Conference on Management of Data*, pages 119–127. ACM, 2022.

[Wik24] Wikipedia. Chernoff bound. https://en.wikipedia.org/wiki/Chernoff_bound, 2024. Accessed: 2024-11-06.