Last compiled October 13, 2025 at 1:57pm.
©Kent Quanrud, 2025.

Randomized Algorithms, CS588, Fall 2025

Lectures: 4:30 to 5:45 PM, Tuesday and Thursday, in Forney Hall G124.

Staff:	Email (`@purdue.edu`)	Office hours
Kent Quanrud	`krq`	Tuesday, 2:30–3:30PM, Lawson 1211
Zilin Shen	`shen624`	Wednesday, 5–6PM, DSAI B055
Tanmay Devale	`tdevale`	Thursday, 2–3PM, DSAI B061

Please see the syllabus (Page 443) for more information.

If you are unable to register for the class because it is full, just wait. Historically, slots have always opened up. You can attend class, add yourself to gradescope, and submit homework in the meantime.

Homework

- Homework 0 due September 4.
- Homework 1 due September 18.
- Homework 2 due October 2.
- Homework 3 due October 16.
- Homework 4 due November 6.
- Homework 5 due November 20.
- Homework 6 due December 4.

Class schedule[1]

1. *August 26.* Randomized SAT and quicksort (Chapter 1). Please read Chapter 0 before class.
2. *August 28.* Hashing and heavy hitters (Chapter 2).
3. *September 2.* Hash tables and linear probing (Chapter 3).
4. *September 4.* Randomized minimum cut (Chapter 4).
5. *September 9.* Random graphs (Chapter 5).
6. *September 11.* Randomized rounding (Chapter 6).

[1]All future dates are tentative. Lecture materials and video recordings can be found at the end of each chapter.

7. *September 16.* Online algorithms (Chapter 7).
8. *September 18.* Distinct elements and moment estimation (Chapter 8).
9. *September 23.* Dimensionality reduction (Chapter 9).
10. *September 25.* Approximate nearest neighbors and LSH. (Chapter 10).
11. *September 30.* Line embeddings and sparsest cut (Chapter 11).
12. *October 2.* Randomized tree metrics (Chapter 12).
13. *October 7.* Sampling geometric range spaces (Chapter 13).
14. *October 9.* PAC learning (Chapter 14).

(⋆) *October 14.* Fall break (October 13–14).

15. *October 16.* Randomized greedy algorithms (Chapter 15).

(⋆) *Saturday, October 18.* Purdue Half-Marathon and 5K

(⋆) *Thursday, October 23, 8–10PM, WTHR 104.* Midterm 1. (See Chapter C for scrambled problems.)

16. *October 28.* Entropy and error-correcting codes (Chapter 16).
17. *October 30.* Lovász local lemma (Chapter 17).
18. *November 4.* Pagerank (Chapter 19).
19. *November 6.* Connectivity and electricity (Chapter 20).

(⋆) *Saturday, November 8.* Monumental Marathon in Indianapolis

20. *November 11.* Spectral analysis of random walks (Chapter 21).
21. *November 13.* Mixing times, conductance, and sparsest cut (Chapter 22).
22. *November 18.* Deterministic log-space connectivity (Chapter 23).
23. *November 20.* Reducing randomness with random walks (Chapter 24).
24. *November 25.* PCP Theorem (Chapter 25).[2]

(⋆) *November 27.* Thanksgiving (November 27)

25. *December 2.* PCP theorem (cont'd).
26. *December 4.* Discrepancy via Gaussian random walks (Chapter 26).
27. *December 9.* Sparsification (Chapter 28).
28. *December 11.* Last day of class. TBD.

(⋆) Final. Wednesday, 7–9PM, Krannert G016

[2]November 25 is the last date to withdraw with a "W" grade.

Contents

Chapter 0

A little probability theory

We need probability theory to understand randomized algorithms. Here is a minimal amount of background to get us started. The reader may have seen these topics before. Subsequent chapters develop probability theory further alongside algorithms and applications.

0.1 Events and probabilities

Suppose we flip a coin. As the coin rotates in mid-air, at its vertical peak, we pause time. Will the coin land heads or tails? Obviously, *we don't know yet.* Yet we state without ambiguity or hesitation that *half the time it will land heads*, and *half the time it will land tails*.

What do we mean when we say that "half the time it will land heads"? There is of course only one coin, and we can't split the coin in half. Rather, we *imagine* that if we repeated the experiment many times, then half the coin tosses *should* come up heads.

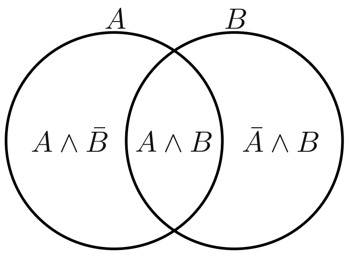
This simple example, which we all understand thoroughly, points to a deeper feature: *probability interprets fractional values as discrete ones.* Here, "half heads" does not mean that "half the coin will come up heads", which is total nonsense; rather it means that half *the time* the coin will come up heads.

The formal rules of probability are simple and intuitive. (The tricky part is sticking to them.) Probability theory assumes an uncertain world where *events* occur with fixed (but not necessarily known) numerical probabilities. Each event A has a probability between 0 and 1, denoted

$$\mathbf{P}[A] \in [0,1].$$

For every event A, there is the complementary event, $\bar{A}$, of A *not occurring.* We always have

$$\mathbf{P}[A] + \mathbf{P}\left[\bar{A}\right] = 1. \tag{1}$$



Joint events. For any two events A and B, one can define the *conjunctive event* that *both A and B* occurs, denoted

$$\text{“}A \wedge B\text{” or “}A \cap B\text{” or “}A \text{ and } B\text{”.}$$

It is common to write $\mathbf{P}[A, B]$ as a shorthand for $\mathbf{P}[A \wedge B]$.

In general, given A and B, there are four disjoint events induced by their combination:

1. $A \wedge B$: The event that both A and B occur.
2. $A \wedge \bar{B}$: The event that A occurs and B does not.
3. $\bar{A} \wedge B$: The event that A does not occur and B does.
4. $\bar{A} \wedge \bar{B}$: The event that neither A nor B occcurs.

The four joint events listed are mutually exclusive – at most one of them can be realized – and exhaustive – at least one of them will be realized. So we have the following identity:

$$\mathbf{P}[A \wedge B] + \mathbf{P}\left[A \wedge \bar{B}\right] + \mathbf{P}\left[\bar{A} \wedge B\right] + \mathbf{P}\left[\bar{A} \wedge \bar{B}\right] = 1$$

Suppose event A occurs with positive probability. Whenever A occurs, then of course exactly one of B occurs or $\bar{B}$ occurs. So we have

$$\mathbf{P}[A] = \mathbf{P}[A, B] + \mathbf{P}\left[A, \bar{B}\right].$$

If we divide both sides by $\mathbf{P}[A]$, we have

$$1 = \frac{\mathbf{P}[A, B]}{\mathbf{P}[A]} + \frac{\mathbf{P}\left[A, \bar{B}\right]}{\mathbf{P}[A]}.$$

This equation looks probabilistic – we have two nonnegative terms summing to 1. The first term on the right-hand side, $\mathbf{P}[A, B] / \mathbf{P}[A]$, can be interpreted as follows:

> *of the times that event A occurs, B also occurs $\mathbf{P}[A, B] / \mathbf{P}[A]$ fraction of the time.*

We call this the *conditional probability of event B conditional on event A*, denoted $\mathbf{P}[B \mid A]$ and defined as the ratio

$$\mathbf{P}[B \mid A] = \frac{\mathbf{P}[A, B]}{\mathbf{P}[A]}.$$

Likewise we have the conditional probabilities $\mathbf{P}\left[\bar{B} \,\middle|\, A\right]$, $\mathbf{P}\left[B \,\middle|\, \bar{A}\right]$, $\mathbf{P}\left[\bar{B} \,\middle|\, \bar{A}\right]$, and so forth.

We always have

$$\mathbf{P}[A \wedge B] \leq \min\{\mathbf{P}[A], \mathbf{P}[B]\}$$

(see exercise C.26). It is generally *not* true that

$$\mathbf{P}[A \mid B] = \mathbf{P}[A] \tag{2}$$

for two events A and B. (2) is equivalent to the same equation with A and B flipped, as well as the identity

$$\mathbf{P}[A \wedge B] = \mathbf{P}[A]\,\mathbf{P}[B] \tag{3}$$

Events A and B are said to be *independent events* in the special case where Eqs. (2) and (3) holds.

Unions of events. We also have the *disjunctive event* that *either A or B* occurs, denoted

$$\text{“}A \vee B\text{” or “}A \cup B\text{” or “}A \text{ or } B\text{”.}$$

If $A \vee B$ occurs, then exactly one of the following events occurs:

$$A \wedge B,\ A \wedge \bar{B},\ \bar{A} \wedge B.$$

Consequently we have

$$\mathbf{P}[A \vee B] = \mathbf{P}[A \wedge B] + \mathbf{P}\left[A \wedge \bar{B}\right] + \mathbf{P}\left[\bar{A} \wedge B\right].$$

Recall that $\mathbf{P}[A] = \mathbf{P}[A \wedge B] + \mathbf{P}\left[A \wedge \bar{B}\right]$, and similarly for $\mathbf{P}[B]$. Adding $\mathbf{P}[A \wedge B]$ to both sides of the identity above gives

$$\mathbf{P}[A \vee B] + \mathbf{P}[A \wedge B] = \mathbf{P}[A] + \mathbf{P}[B].$$

This identity reflects a venn-diagram, so to speak, where the two regions A and B “sum” to their union $A \vee B$ and their intersection $A \wedge B$.

Dropping the nonnegative term $\mathbf{P}[A \wedge B]$ from the identity above gives rise to the following extremely useful *union bound*.

Lemma 0.1 (Union bound)**.** *For any two events A and B,*

$$\mathbf{P}[A \vee B] \leq \mathbf{P}[A] + \mathbf{P}[B],$$

with equality iff $\mathbf{P}[A \wedge B] = 0$.

0.2 Random variables

A *finite random variable* models an unrealized and uncertain object X that takes one of a finite set of values, $\{x_1, \dots, x_k\}$.[1] For each outcome x_i, "X equals x_i" is an event, with a fixed probability, denoted $\mathbf{P}[X = x_i]$. These probabilities sum to 1:

$$\sum_{i=1}^{k} \mathbf{P}[X = x_i] = 1.$$

For example, we can describe a coin toss as a random variable $X \in \{\texttt{heads}, \texttt{tails}\}$. If the coin comes up heads, then $X = \texttt{heads}$. If the coin comes up tails, then $X = \texttt{tails}$. For a fair coin we have

$$\mathbf{P}[X = \texttt{heads}] = \mathbf{P}[X = \texttt{tails}] = \frac{1}{2}.$$

Note that these two probabilities sum to 1.

If we have two random variables $X \in \{x_1, \dots, x_k\}$ and $Y \in \{y_1, \dots, y_\ell\}$, then their product (X, Y) forms a random variable in the set $\{(x_i, y_j) : i = 1, \dots, k,\ j = 1, \dots, \ell\}$. We have probabilities of the form

$$\mathbf{P}[X = x_i,\ Y = Y_j].$$

It is not generally true

$$\mathbf{P}[X = x_i,\ Y = y_k] = \mathbf{P}[X = x_i]\,\mathbf{P}[Y = y_k]$$

When the equation above holds for all x_i and y_j, then X and Y are said to be *independent*. This is equivalent to saying that for all x_i and y_j, the events $X = x_i$ and $Y = y_j$ are independent.

[1]One can also define continuous variable (e.g., that take values continuously between 0 and 1), where sums are replaced by integrals.

For example, suppose $X, Y \in \{\mathsf{heads}, \mathsf{tails}\}$ both describe coin tosses. If they described different coin tosses, then they would be independent random variables, and each combination of heads and tails would occur with probability .25. That is,

$$\begin{aligned}&\mathbf{P}[X = \mathsf{heads}, Y = \mathsf{heads}] = \mathbf{P}[X = \mathsf{heads}, Y = \mathtt{tails}]\\&= \mathbf{P}[X = \mathtt{tails}, Y = \mathsf{heads}] = \mathbf{P}[X = \mathtt{tails}, Y = \mathtt{tails}] = \frac{1}{4},\end{aligned}$$

Thus X and Y are independent random variables. If they modeled the *same coin*, then we would have

$$\mathbf{P}[X = \mathsf{heads}, Y = \mathsf{heads}] = \mathbf{P}[X = \mathtt{tails}, Y = \mathtt{tails}] = \frac{1}{2},$$

while

$$\mathbf{P}[X = \mathsf{heads}, Y = \mathtt{tails}] = \mathbf{P}[X = \mathtt{tails}, Y = \mathsf{heads}] = 0.$$

Here, X and Y are *not* independent.

0.3 Averages

When a random variable X takes on real values, we can have a well-defined and quantitative notion of an "average", more formally called the *expected value.*

Definition 0.2. *Let $X \in \mathbb{R}$ be a real-valued random variable that has a finite set of possible values. Then the* expected value of X*, denoted* $\mathbf{E}[X]$*, is the weighted sum*

$$\mathbf{E}[X] \overset{\text{def}}{=} \sum_x \mathbf{P}[X = x] \cdot x;$$

where the sum is over all values of x where $\mathbf{P}[X = x] > 0$.

For continuous random variables, the sum would be replaced by an integral.

The average quantity of a random variable is very intuitive; the reader is likely used to discussing averages in the sense defined above. (e.g., the *average* midterm score.) The following identity, called *linearity of expectation*, is perhaps less intuitive; however it follows rather plainly from the definition of expectation.

Theorem 0.3. (Linearity of expectation.) *Let $X, Y \in \mathbb{R}$ be two random variables. Then*

$$\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y].$$

The proof of linearity of expectation is given as exercise C.46. The reader may want to first consider the simple case where $X \in \{x_1, x_2\}$ takes on exactly two values, and $Y \in \{y_1, y_2\}$ takes on exactly two values. One can generalize to finite sets from there.

Observe that linearity of expectation does not make any assumptions about how X and Y are structured or related. This makes linearity of expectation extremely useful and often leads to surprising observations.

A simple example of linearity of expectation is as follows. Consider a population of people with various heights. Let X and Y be two quantities obtained by the following experiment. Draw one person uniformly at random. Let X be the length from the waist of this person to the top of their head. Let Y be the length from the waist of this person to the ground. $X + Y$ gives to the total height of the person. Note that X and Y are highly dependent, since they both measure the same (randomly drawn) person. Linearity of expectation says:

$$\underbrace{(\text{average height})}_{\mathbf{E}[X+Y]} = \underbrace{\begin{pmatrix}\text{average length}\\ \text{from waist up}\end{pmatrix}}_{\mathbf{E}[X]} + \underbrace{\begin{pmatrix}\text{average length}\\ \text{from waist down}\end{pmatrix}}_{\mathbf{E}[Y]}.$$

Of course, this makes total sense.

Conditional expected values. We also have conditional expected value analogous to how we have conditional probabilities. For a random variable X and an event A, $\mathbf{E}[X \mid A]$ denotes the expected value of X conditional on A occurring. Formally we have

$$\mathbf{E}[X \mid A] = \sum_x x\, \mathbf{P}[X = x \mid A].$$

The reader should verify that

$$\mathbf{E}[X] = \mathbf{E}[X \mid A]\, \mathbf{P}[A] + \mathbf{E}\left[X \,\middle|\, \bar{A}\right] \mathbf{P}\left[\bar{A}\right]$$

for all events A.

Now, suppose we have two random variables X and Y, and a real-valued function $f(X, Y)$ of these variables. Then by definition we have

$$\mathop{\mathbf{E}}_{X,Y}[f(X,Y)] = \sum_{(x,y)} f(x,y)\, \mathbf{P}[X = x, Y = y]. \tag{4}$$

Here we have annotated X, Y under the $\mathbf{E}[\cdots]$ to emphasize that the randomization is over X and Y jointly. On the other hand consider the following *nested* expected value,

$$\mathop{\mathbf{E}}_{X}\left[\mathop{\mathbf{E}}_{Y}[f(X,Y)\,|\,X]\right].$$

This version describes the average of an experiment where we first observe X, and then conditional on X, we observe Y and evaluate $f(X,Y)$. Formally the expression expands out to

$$\begin{aligned}\mathop{\mathbf{E}}_{X}\left[\mathop{\mathbf{E}}_{Y}[f(X,Y)\,|\,X]\right] &= \sum_{x}\mathop{\mathbf{E}}_{Y}[f(X,Y)\,|\,X=x]\,\mathbf{P}[X=x] \\ &= \sum_{x}\sum_{y} f(x,y)\,\mathbf{P}[Y=y\,|\,X=x]\,\mathbf{P}[X=x]. \qquad (5)\end{aligned}$$

Since $\mathbf{P}[X=x,Y=y]=\mathbf{P}[Y=y\,|\,X=x]\,\mathbf{P}[X=x]$, the RHSs of Eqs. (4) and (5) are equal, hence

$$\mathop{\mathbf{E}}_{X,Y}[f(X,Y)] = \mathop{\mathbf{E}}_{X}\left[\mathop{\mathbf{E}}_{Y}[f(X,Y)\,|\,X]\right].$$

(See also exercise C.3.)

A very nice trick, called the *law of iterated expectations*, is useful in the following situation. Suppose we had a random variable X for which we want to evaluate $\mathbf{E}[X]$. Suppose it is hard to analyze $\mathbf{E}[X]$ directly, but for some contextual reason there is a second random variable Y for which the conditional expectation $\mathbf{E}[X\,|\,Y]$ is better understood. Then $\mathbf{E}[X]$ might be computed indirectly via

$$\mathop{\mathbf{E}}_{X}[X] = \mathop{\mathbf{E}}_{X,Y}[X] = \mathop{\mathbf{E}}_{Y}\left[\mathop{\mathbf{E}}_{X}[X\,|\,Y]\right].$$

For a simple example (found on the internet), suppose we wanted to estimate the probability that it rains tomorrow. Thus let $X=1$ if it rains tomorrow and 0 if not; $\mathbf{E}[X]$ is the probability that it rains. Suppose we also know the probability that it will rain today, as well as:

1. the probability of it raining tomorrow if it rains today, and
2. the probability of it raining tomorrow if it does not rain today.

Let $Y=1$ if it rains today and 0 if not; in terms of X and Y, we are assuming that know $\mathbf{E}[Y]=\mathbf{P}[Y=1]$, $\mathbf{E}[X\,|\,Y=1]$, and $\mathbf{E}[X\,|\,Y=0]$. Then we can obtain the probability of it raining tomorrow, $\mathbf{E}[X]$, via the law of iteration expectations, as

$$\mathbf{E}[X] = \mathop{\mathbf{E}}_{Y}\left[\mathop{\mathbf{E}}_{X}[X\,|\,Y]\right] = \mathbf{P}[Y=1]\,\mathbf{E}[X\,|\,Y=1] + \mathbf{P}[Y=0]\,\mathbf{E}[X\,|\,Y=0].$$

Markov's inequality. Markov's inequality is about how much a nonnegative random variable can greatly exceed its expectation.

Lemma 0.4 (Markov's inequality)**.** *Let $X \geq 0$ be a* nonnegative *random variable, and $\alpha \geq 0$. Then*

$$\mathbf{P}[X \geq \alpha] \leq \mathbf{E}[X]/\alpha.$$

Proof. Since X is nonnegative we have

$$\mathbf{E}[X] = \mathbf{E}[X \mid X \geq \alpha]\,\mathbf{P}[X \geq \alpha] + \mathbf{E}[X \mid X < \alpha]\,\mathbf{P}[X < \alpha] \geq \alpha\,\mathbf{P}[X \geq \alpha]$$

for all $\alpha \geq 0$. □

Markov's inequality is very intuitive. Consider for example $\alpha = 2\,\mathbf{E}[X]$. Then Markov's inequality states that for nonnegative X, the probability that X is at least twice its average is at most 50%. This should be as obvious as the fact that no more than half the population can be twice as wealthy as the average individual. Or that no more than half the class can get at least twice the average score on the midterm, no matter how low the average. Etc.

0.4 Exercises

Additional exercises may be found in [MR95, Chapter 1].

Exercise 0.1. Prove or disprove: For any two events A, B,

$$\mathbf{P}[A \wedge B] \leq \min\{\mathbf{P}[A], \mathbf{P}[B]\}.$$

Exercise 0.2. Prove or disprove: For any two events A and B, if $\mathbf{P}[A] + \mathbf{P}[B] > 1$, then $\mathbf{P}[A \wedge B] > 0$.

Exercise 0.3. Let A and B be two events. Prove that the following three identities are all equivalent:

$$\mathbf{P}[A \mid B] = \mathbf{P}[A], \qquad \mathbf{P}[B \mid A] = \mathbf{P}[B], \qquad \mathbf{P}[A, B] = \mathbf{P}[A]\,\mathbf{P}[B].$$

(That is, if A and B satisfies any one of the identities above, then it automatically satisfies the other two.)

Exercise 0.4. Let A and B two events. Prove or disprove that A and B are independent iff $\bar{A}$ and $\bar{B}$ are independent.

Exercise 0.5. Prove linearity of expectation (Theorem 0.3).

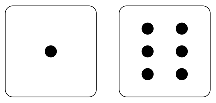
Exercise 0.6. Prove or disprove: for any two random variables X, Y, and real-valued function $f(X,Y)$, we have

$$\underset{X}{\mathbf{E}}\left[\underset{Y}{\mathbf{E}}[f(X,Y) \mid X]\right] = \underset{Y}{\mathbf{E}}\left[\underset{X}{\mathbf{E}}[f(X,Y) \mid Y]\right].$$

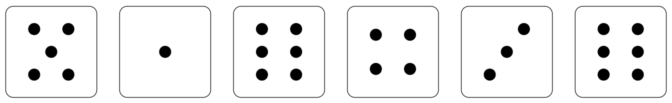
Exercise 0.7. Let A and B be two events with $\mathbf{P}[A] + \mathbf{P}[B] = 1$. Prove or disprove: $\mathbf{P}[A \vee B] = 1$ iff $\mathbf{P}[A \wedge B] = 0$.

Exercise 0.8. Suppose you only have access to a coin that flips heads with a known probability p, and tails with (remaining) probability $1 - p$. Describe and analyze a protocol that uses a limited number of tosses of this biased coin in expectation (the smaller the better) to simulate 1 coin toss of a fair coin. (The expected number of biased coin tosses you make may depend on p.)

Exercise 0.9. Recall that when we roll six-sided dice, the dice samples an integer between 1 and 6 uniformly at random. Let us call an unordered pair of dice "lucky" if one of them is a 1 and the other is a 6.



If we roll 6 independent six-sided dice, how many lucky pairs do we expect? Note that a single dice may appear in more than one lucky pair. For example, the following roll of six dice has 2 lucky pairs amongst them.



Exercise 0.10. Suppose you repeatedly flip a coin that is heads with fixed probability $p \in (0, 1)$.

1. What is the expected number of coin flips until you obtain one heads?[2] Prove your answer.[3]

2. What is the expected number of coin flips until you obtain two heads? Prove your answer.

3. For general $k \in \mathbb{N}$, what is the expected number of coin tosses until you obtain k heads? Prove your answer.

Exercise 0.11. Let X, Y be two independent and identically distributed real random variables. Prove that $\mathbf{P}[|X - Y| \leq 2] \leq 3\,\mathbf{P}[||X - Y| \leq 1|]$.

[2]If the first toss is heads, that counts as one coin flip. If the first toss is tails and the second toss is heads, that counts as two coin tosses. Etc. It may be helpful to first think about a fair coin, where $p = 1/2$.

[3]*Hint:* There is an easy way and a hard way to solve this problem.

Chapter 1

Randomized searching and sorting

1.1 First randomized algorithms

We introduce randomized algorithms via 3 basic problems: 3-SAT, sorting, and selection. For each we present randomized algorithms that are essentially optimal (if one does not mind that they are randomized). They are also very simple to describe and code. They might seem tricky to analyze, as the analysis involves probabilities and sorting through nondeterministic outcomes. We will see how focusing on *expected* performance renders the analysis surprisingly clean.

Let us briefly introduce the algorithms first as their striking simplicity may help motivate us to learn the mathematical tools we need to analyze them. We first present the algorithms, all strikingly simple, and then we will analyzing each one separately.

1.1.1 3-SAT.

We start with 3-SAT. The input to 3-SAT consists of a Boolean formula in conjunctive normal form (CNF) with 3 (distinct) variables per clause. For example,

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$$

is a 3-SAT formula with $m = 4$ clauses and $n = 3$ variables. It is satisfied by the assignment $(x_1, x_2, x_3) = (\texttt{t}, \texttt{f}, \texttt{t})$. ($\texttt{t}$ denotes "true" and $\texttt{f}$ denotes "false".) In the 3-SAT problem, we are given a 3-SAT formula $f(x_1, \ldots, x_n)$ with m clauses and n variables and want to find a satisfying assignment.

3-SAT is the quintessential NP-Complete search problem, and there is no polynomial time algorithm to solve it unless $P = NP$ [Coo71; Lev73]. But this does not prevent us from trying to *approximate* the problem. The goal is now to find an assignment that satisfies as many clauses as possible. Of course an exact algorithm for this maximization version implies a polynomial time algorithm for the decision

```
random-SAT($f(x_1,\dots,x_n)$)
1. For each $i \in [n]$, draw $x_i \in \{t, f\}$ independently and uniformly at random.
2. Return $x_1,\dots,x_n$.
```

Figure 1.1: A randomized approximation algorithm for SAT.

version. Instead we will design algorithms that do not guarantee the maximum, but are competitive up to a multiplicative factor when compared to the optimum solution.

Given a SAT formula f, let OPT denote the maximum number of clauses that are satisfiable. For $\alpha \in [0,1]$, an *α-approximation algorithm* for SAT is an algorithm that produces an assignment that satisfies at least $\alpha\,\mathrm{OPT}$ clauses. While obtaining an exact algorithm is NP-Hard, for fixed $\alpha < 1$, it is not necessarily NP-Hard to obtain an α-approximation algorithm for SAT.

Here is a simple randomized approximation algorithm for SAT. Given a formula $f(x_1,\dots,x_n)$, for each variable x_i, flip a fair coin and assign $x_i = \mathtt{t}$ or $x_i = \mathtt{f}$ accordingly. (See Fig. 1.1 for pseudocode.) We will show that, on average, this random assignment satisfies at least $(7/8)m$ clauses out of m total. Moreover, we will be able to *derandomize* the above algorithm and obtain a deterministic algorithm that (always) satisfies at least $(7/8)m$ clauses.

The randomized algorithm is extremely simple and basically oblivious to the input. Surely it isn't very good. But in fact it is the best possible polynomial time algorithm unless $P = NP$. The *PCP theorem* states that for all constants $\epsilon > 0$, getting better than a $(7/8 + \epsilon)$-approximation to 3SAT is NP-Hard. "PCP" standards for *probabilistically checkable proofs*. The PCP theorem gives similar *hardness of approximation* results for many other problems besides SAT. The PCP theorem (as the name suggests) has strong connections to randomized algorithms. We will discuss related randomized topics, and maybe parts of the proof of the PCP theorem, later in the course.

Thus in this lecture we will show the first part of the following theorem. Topics in later chapters will hopefully shed some light on the second half of the theorem.

Theorem 1.1. *There is a polynomial time algorithm that given any 3-SAT formula computes an assignment that satisfies at least $\frac{7}{8}$th of the clauses. Moreover, for all $\epsilon > 0$, a polynomial time approximation algorithm with approximation ratio $\left(\frac{7}{8} + \epsilon\right)$ implies that $P = NP$.*

```
quick-sort($A[1..n]$)
/* For simplicity we assume all the elements are distinct. Otherwise, break ties consistently. */
1. If $n \leq 1$ then return $A$.
2. Select $i \in [n]$ uniformly at random.
3. $B[1..k] \leftarrow$ recursively sort the set of elements less than $A[i]$.
4. $C[1..\ell] \leftarrow$ recursively sort the set of elements greater than $A[i]$.
5. Return the concatenation of $B$, $A[i]$, and $C$.
```

Figure 1.2: A randomized sorting algorithm.

1.1.2 Sorting.

The next problem we discuss is sorting. The goal is to take an unordered list of n comparable elements (e.g., numbers) and return them as a list in sorted order. The reader likely knows that the `merge-sort` algorithm runs in $O(n \log n)$, and that there is a $\Omega(n \log n)$-time lower bound for any sorting algorithm in the comparison model. Here we will study a randomized algorithm that is remarkably simple, called `quick-sort`, that is often the preferred one in practice. The idea is very simple: select an element *uniformly at random* out of the list to serve as a *pivot*. Divide the elements into those smaller and larger than the pivot, and recurse on both halves. See Fig. 1.2 for pseudocode.

What is the worst-case running time of `quick-sort`? It is important to clarify what we mean by "worst-case". Observe that the running time of `quick-sort` is proportional to the total number of comparisons made by the algorithm. It is certainly possible that the algorithm makes $\Omega(n^2)$ comparisons. (How?) So in a limited sense that algorithm has a worst-case $O(n^2)$ time.

As the algorithm is randomized, a more useful measure is the *average* number of comparisons. We will show that `quick-sort` takes $O(n \log n)$ time on average *against any input*. This is still a worst-case analysis in the sense that it holds for *all inputs*. (This is not to be confused with the performance of an algorithm against a *randomized input from a fixed distribution* — that is called *average case analysis*.) We will also show that the algorithm takes $O(n \log n)$ time with extremely high probability. We prove, in Section 1.3:

Theorem 1.2. *Given a list of n comparable elements,* `quick-sort` *returns the elements in a sorted list in $O(n \log n)$ expected time and with high probability.*

```
quick-select(A[1..n],k)
/* The goal is to find the rank k element in A[1..n]. We assume for simplicity that all the
   elements are distinct.                                                                  */
1. Randomly select $i \in [n]$ uniformly at random.
2. Compute the rank $\ell$ of $A[i]$.                                                  // $O(n)$
3. If $\ell = k$, then return $A[i]$.
4. If $\ell > k$, then recursively search for the rank $k$ element among the set of $\ell - 1$ elements less
   than $A[\ell]$, and return it.
5. If $\ell < k$, then recursively search for the rank $k - \ell$ element among the set of $n - \ell$ elements
   greater than $A[\ell]$, and return it.
```

Figure 1.3: A randomized algorithm for selection.

1.1.3 Selection.

The last problem we mention is selection. The input, similar to sorting, includes an unordered list of n comparable elements. Given an index $k \in [n]$, the goal is to find the kth smallest element in the list.

The obvious solution is to sort the list, which takes $O(n \log n)$ time. Famously, one can do better: the "median-of-medians" divide-and-conquer algorithm of Blum, Floyd, Pratt, Rivest, and Tarjan [BFP+72] runs in $O(n)$ time. This algorithm is a bit tricky, both to describe and to analyze. Here then is a simpler alternative, which is similar to quick-sort: pick a pivot uniformly at random, and compute its rank ℓ. Depending on whether $k = \ell$, $k < \ell$, or $k > \ell$, either return the pivot, recurse on the subset of elements less than the pivot, or recurse on the subset greater than the pivot. See Fig. 1.3 for the pseudocode.

We will prove the following theorem which states that `quick-select` takes $O(n)$ time in expectation. Or rather, *you* will prove it, in exercise C.31, employing the new tools gained from analyzing randomized SAT and sorting.

Theorem 1.3. `quick-select(`$A[1..n]$`,`k`)` *returns the rank k element in $O(n)$ time in expectation and with high probability.*

1.2 Randomized approximations for SAT

Recall the very simple algorithm (Fig. 1.1) we want to analyze: given a 3-SAT formula $f(x_1, \ldots, x_n)$, independently flip a fair coin for each x_i and assign $x_i \in$

$\{\mathtt{t}, \mathtt{f}\}$ accordingly. We claim that this algorithm gives a (7/8)th approximation (in expectation).

Consider a single clause; e.g., $(x_1 \vee \bar{x}_2 \vee x_3)$. Of all 8 ways to assign the three variables (x_1, x_2, x_3 in the example) values in $\{\mathtt{t}, \mathtt{f}\}$, there is only one way that does *not* satisfy the clause. That is,

each clause is satisfied with probability $7/8$.

Simple enough. However we are interested not in the outcome of a single clause, but the total number of clauses that are satisfied. That is to say we are analyzing many clauses and not just one. As a warm up, suppose we had two clauses, say C_1 and C_2. Let E_1 (resp. E_2) denote the event that C_1 (resp. C_2) is satisfied. Here the analysis varies depending on whether C_1 and C_2 share variables.

In the simplest case, suppose C_1 and C_2 have no variables in common. Then C_1 and C_2 depend on completely different coin tosses, so the events E_1 and E_2 are independent. Thus, the probability that they are both satisfied is

$$\mathbf{P}[E_1, E_2] = \mathbf{P}[E_1]\,\mathbf{P}[E_2] = \left(\frac{7}{8}\right)^2.$$

Likewise one can obtain probabilities that exactly one clause, or neither clause, is satisfied, arriving at the following table of joint probabilities.

$\mathbf{P}[\cdot,\cdot]$	E_2	$\bar{E}_2$
E_1	49/64	7/64
$\bar{E}_1$	7/64	1/64

By direct calculation[1] one will find that 7/4 clauses are satisfied (out of a maximum of 2) on average.

The calculations above were clean insofar as C_1 and C_2 are disjoint. What if they shared variables? Suppose for example they both had a variable x_3 in common; say,

$$C_1 = (x_1 \vee \bar{x}_2 \vee x_3) \text{ and } C_2 = (x_3 \vee x_4 \vee \bar{x}_5).$$

Now x_3 has greater importance: if $x_3 = \mathtt{t}$, then both C_1 and C_2 are immediately satisfied; if tails, then both C_1 and C_2, independently, need one of two coins to come up in their favor to be satisfied.

It is helpful in this example to map out the scenarios depending on the outcome of x_3, via *conditional expectations*. (We can *imagine* x_3 is flipped first, even if it is actually not.) For example, for $E_1 \wedge E_2$, we have

$$\mathbf{P}[E_1, E_2] = \mathbf{P}[E_1, E_2 \,|\, x_3 = \mathtt{t}]\,\mathbf{P}[x_3 = \mathtt{t}] + \mathbf{P}[E_1, E_2 \,|\, x_3 = \mathtt{f}]\,\mathbf{P}[x_3 = \mathtt{f}].$$

[1] $2 \cdot (49/64) + 7/64 + 7/64 = 7/4.$

Now, we have $\mathbf{P}[x_3 = \mathtt{t}] = \mathbf{P}[x_3 = \mathtt{f}] = 1/2$. If $x_3 = \mathtt{t}$, then $E_1 \wedge E_2$ always occurs. Now consider the case where $x_3 = \mathtt{f}$. Since C_1 and C_2 have no overlap besides x_3, the events E_1 and E_2, conditional on $x_3 = \mathtt{f}$, are now independent. Thus

$$\mathbf{P}[E_1, E_2 \mid x_3 = \mathtt{f}] = \mathbf{P}[E_1 \mid x_3 = \mathtt{f}]\, \mathbf{P}[E_2 \mid x_3 = \mathtt{f}] = \left(\frac{3}{4}\right)^2.$$

Altogether we obtain

$$\mathbf{P}[E_1, E_2] = 1 \cdot \frac{1}{2} + \left(\frac{3}{4}\right)^2 \cdot \frac{1}{2} = 25/36.$$

Likewise, one can compute remaining values $\mathbf{P}\left[E_1 \wedge \bar{E}_2\right]$, $\mathbf{P}\left[\bar{E}_1 \wedge \bar{E}_2\right]$, and $\mathbf{P}\left[\bar{E}_1 \wedge \bar{E}_2\right]$ from the 2×2 table. Then by direct calculation one obtains the expected number of clauses that are satisfied in this scenario. If the reader works out the calculations they will find that the expected number is (again) 7/4.

Besides the scenarios above, we can also have the cases where C_1 and C_2 overlap at 2 variables, or all 3. We also need to consider scenarios where a variable x appears in C_1 while its negation $\bar{x}$ appears in C_2. Each of these situations require another series of calculations. The point is that its getting messier and messier, even with just two clauses.

Now imagine trying to analyze three clauses C_1, C_2, C_3. The number of ways the three clauses can relate grows combinatorially. In general, over m clauses $C_1, \ldots, C_m$, there are just too many possible scenarios to calculate everything exactly. The more general question is: *how do we precisely analyze a randomized mechanism that encompasses a combinatorial explosion of possibilities?*

The key insight is that we are only interested in the *average* number of clauses satisfied. Observe that the average, being a reductive aggregate statistic, does not reveal much information about the fine-grained complexities about the clauses. Perhaps all these details are not necessary to assess the average either.

For each clause C_i, let $Y_i \in \{0, 1\}$ be the random variable indicating whether or not C_i is satisfied:

$$Y_i = \begin{cases} 1 & \text{if } C_i \text{ is satisfied,} \\ 0 & \text{otherwise.} \end{cases}$$

Thus $\sum_{i=1}^m Y_i$ is the number of clauses satisfied, and we seek the quantity $\mathbf{E}[\sum_{i=1}^m Y_i]$.

As established above, it is easy to see that

$$\mathbf{E}[Y_i] = 7/8$$

for each i. The difficulty is in understanding how to analyze the sum of the Y_i's *jointly* as they may be strongly connected with each other. Thank goodness, then, for *linearity of expectation: the expected sum equals the sum of expectations.* By linearity of expectation, we have

$$\mathbf{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbf{E}[Y_i] = \frac{7}{8}m.$$

In the final analysis we pay no attention to the intricate relationships among the C_i's. Such is the power of linearity of expectations, to help us see in the aggregate what is far too complicated to understand in detail.

It is easy to extend the above to k-SAT for any $k \in \mathbb{N}$, where each clause has exactly k variables, and obtain the following.

Theorem 1.4. *For all $k \in \mathbb{N}$, there is a randomized $(1 - 1/2^k)$-approximation algorithm for k-SAT.*

Derandomization. Its hard to imagine a more random algorithm than flipping a coin for each variable. Can we obtain a $(7/8)$-approximation factor *deterministically*? The answer is yes, and perhaps more surprisingly, we will use the randomized algorithm as a guide.

As before, for each clause C_i, let Y_i indicate whether or not a clause is satisfied. Let $Z = \sum_{i=1}^{m} Y_i$. When all the variables are assigned uniformly at random, $\mathbf{E}[Z] = (7/8)m$.

Among many other decisions, a deterministic algorithm must decide whether to set $x_1 = \mathtt{t}$ or $x_1 = \mathtt{f}$. We already know that a uniformly random choice is pretty good on average. Generally speaking, if an average of choices is good, then at least one of those choices ought to be good as well. We just need a way to distinguish the better choice.

Let us be more precise. Suppose all the x_i's are assigned values independently and uniformly at random. By conditional expectations we have

$$\mathbf{E}[Z] = \frac{1}{2}\mathbf{E}[Z \mid x_1 = \mathtt{t}] + \frac{1}{2}\mathbf{E}[Z \mid x_1 = \mathtt{f}].$$

Rearranging, we have

$$\max\{\mathbf{E}[Z \mid x_1 = \mathtt{t}], \mathbf{E}[Z \mid x_1 = \mathtt{f}]\} \geq \frac{1}{2}(\mathbf{E}[Z \mid x_1 = \mathtt{t}] + \mathbf{E}[Z \mid x_1 = \mathtt{f}]) \geq \mathbf{E}[Z].$$

That is, fixing either $x_1 = \mathtt{t}$ or $x_1 = \mathtt{f}$, and then randomly flipping the coins for the remaining variables, will preserve the expected number of satisfied clauses.

To decide which choice is better, we need to be able to compute the conditional expectations $\mathbf{E}[Z \mid x_1 = \mathtt{t}]$ and $\mathbf{E}[Z \mid x_1 = \mathtt{f}]$. Fortunately this is easy to do. Consider the case $x_1 = \mathtt{t}$. By linearity of expectation we have

$$\mathbf{E}[Z \mid x_1 = \mathtt{t}] = \sum_{i=1}^{m} \mathbf{E}[Y_i \mid x_1 = \mathtt{t}].$$

For a particular Y_i, we have

1. $\mathbf{E}[Y_i \mid x_1 = \mathtt{t}] = 1$ if x_i appears (unnegated) in the ith clause, and
2. $\mathbf{E}[Y_i \mid x_1 = \mathtt{t}] = 1 - 2^{-k_i}$ otherwise, where k_i is the number of undecided variables remaining.[2]

Thus we can calculate $\mathbf{E}[Z \mid x_1 = \mathtt{t}]$ and similarly $\mathbf{E}[Z \mid x_1 = \mathtt{f}]$ exactly, and identify the choice of x_1 maximizing the conditional expectation.

Fixing x_1 to be this value from now on, we can now identify the best choice for x_2 in the same way, and continue in such a fashion to make deterministic choices for all x_i's. To make this precise, suppose we have already identified values $a_1, \dots, a_k \in \{\mathtt{t}, \mathtt{f}\}$ for $x_1, \dots, x_k$, respectively, such that

$$\mathbf{E}[Z \mid x_1 = a_1, \dots, x_k = a_k] \geq (7/8)m.$$

Consider x_{k+1}. We have

$$\begin{aligned}\mathbf{E}[Z \mid x_1 = a_1, \dots, x_k = a_k] = \frac{1}{2}\,&\mathbf{E}[Z \mid x_1 = a_1, \dots, x_k = a_k, x_{k+1} = \mathtt{t}] \\ &+ \frac{1}{2}\,\mathbf{E}[Z \mid x_1 = a_1, \dots, x_k = a_k, x_{k+1} = \mathtt{f}],\end{aligned}$$

hence setting either $x_{k+1} = \mathtt{t}$ or $x_{k+1} = \mathtt{f}$ preserves the expected value of Z. We can compute both conditional expectations explicitly, and identify the better choice, setting this value to a_{k+1} accordingly. Continuing in this fashion, we will eventually identify n values $a_1, \dots, a_n$ such that

$$f(a_1, \dots, a_n) = \mathbf{E}[Z \mid x_1 = a_1, \dots, x_n = a_n] \geq (7/8)m,$$

as desired.

The algorithm we have described is entirely deterministic. It uses the estimates of imagined randomized experiments to make its deterministic choices, and the estimates can calculated deterministically thanks in part to linearity of expectation.

[2]For 3-SAT, $k_i = 2$ if $\bar{x}_i \in C_i$, and $k_i = 3$ otherwise.

1.3 Randomized sorting

We now move on to sorting. Recall the simple `quick-sort` algorithm from the introduction, and given in Fig. 1.2. To recap, `quick-sort` is a recursive algorithm where, for each subproblem, we select one of the elements uniformly at random as a pivot. We divide the remaining elements into those that are smaller and greater than the pivot, and recursively sort both groups.

Intuitively, we are hoping that each pivot roughly divides the input in half. If that were always the case, then we would have a $O(n \log n)$ running time by the usual divide-and-conquer analysis. However the random pivot may be bad and break the input into extremely uneven parts, in which case we have made little progress from a divide-and-conquer point of view.

It appears difficult to analyze the running time when progress varies wildly on random choices. It gets more convoluted when one thinks about how a good or bad pivot early effects all the running times thereafter. More generally, one difficulty in analyzing a randomized *algorithm* is that sequences of random decisions generate overwhelmingly many "butterfly effects" to consider.

Fortunately we are not trying to map out all the probabilistic outcomes with complete precision. We are only interested in analyzing the running time on *average*. We will leverage linearity of expectation to greatly simplify the analysis for the following theorem.

Theorem 1.5. *Given a list of n comparable elements,* `quick-sort` *returns elements in a sorted list in $O(n \log n)$ expected time.*

Proof. For each $i, j \in [n]$ with $i < j$, let X_{ij} be equal to 1 if the rank i element (i.e., the ith smallest element) is compared to the rank j element, and 0 otherwise. $\sum_{i<j} X_{ij}$ represents the total number of comparisons made by the algorithm, hence the running time up to a constant factor. We want to upper-bound $\mathbf{E}\left[\sum_{i<j} X_{ij}\right]$.

Consider X_{ij} for fixed $i < j$. Observe that the rank i and rank j numbers are compared to each other iff either is selected as the pivot before any element of rank between i and j. Since the pivots are selected uniformly at random, this occurs with probability $2/(j - i + 1)$. That is,

$$\mathbf{E}[X_{ij}] = \mathbf{P}[X_{ij} = 1] = \frac{2}{j - i + 1}.$$

Consider now the sum $\sum_{i<j} X_{ij}$. While each X_{ij} was simple to analyze alone, the different X_{ij}'s are not at all independent. Fortunately we do not need to map out

their myriad interactions; we are only interested in the X_{ij}'s *in the aggregate* and *on average*. Enter *linearity of expectation*. We have

$$\mathbf{E}\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right] \overset{\text{(a)}}{=} \sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \overset{\text{(b)}}{=} \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n}\sum_{k=1}^{n-i} \frac{2}{k} \leq O(n\log n),$$

as desired. Here (a) applies linearity of expectation: the average sum is equal to the sum of averages. (b) is from our analysis for a single X_{ij} above. □

Bounding the running time with high probability. We've now shown that `quick-sort` terminates in $O(n\log n)$ time *on average*. Next we will show that the running time is at most $O(n\log n)$ with *high probability*. Here, "high probability" means that the probability of error is at most $1/n^c$ for some fixed constant $c > 0$; that is, polynomially small in the input size. (Below we will prove a probability of error of $1/n^2$, but the exponent could have been made arbitrarily large by increasing the hidden constant in the $O(n\log n)$ running time.)

Theorem 1.6. `quick-sort` *runs in $O(n\log n)$ time with high probability.*

Proof. Fix an element e. This element e will appear in a series of recursive subproblems until e is selected as a pivot. Let the *depth* of e, denoted D_e, be the number of recursive subproblems containing e before reaching the base case. Observe that the sum of depths over all the elements, $\sum_e D_e$, bounds the number of comparisons made by the algorithm.

Fix e. We will show that with high probability, D_e is at most $O(\log n)$. More precisely, we will prove that

$$\mathbf{P}[D_e \geq 32\ln n] \leq 1/n^3. \tag{1.1}$$

Assuming (1.1) holds for any e, we then have

$$\mathbf{P}\left[\max_e D_e \geq 32\ln n\right] \overset{\text{(a)}}{\leq} \sum_e \mathbf{P}[D_e \geq 32\ln n] \leq n\cdot\frac{1}{n^3} = \frac{1}{n^2}$$

by (a) the union bound. This establishes that with high probability, *all* elements have depth $O(\log n)$. In this event the running time is $O(\sum_e D_e) = O(n\log n)$, as desired.

It remains to prove (1.1) for a fixed element e. For $i = 0, 1, 2, \ldots,$ let X_i be the number of elements in the subproblem containing e after i recursive calls. Here $X_0 = n$ since initially there are n elements. For depths i after e is selected as a pivot we set $X_i = 0$. Thus $D_e \geq i$ only if $X_i \geq 1$.

We want to upper bound $\mathbf{E}[X_i]$ for each i (and eventually show that $\mathbf{E}[X_i] = 1/\operatorname{poly}(n)$ for $i = O(\log n)$). As mentioned above, $X_0 = n$. Consider X_1. An exact estimate for $\mathbf{E}[X_1]$ is somewhat involved as it depends on the rank of e. A lazier upper bound can be obtained as follows.

For a given subproblem of k elements, call a pivot "good" if it is one of the middle $k/2$ elements, and otherwise "bad". A pivot is good with probability $1/2$, and separates e from at least $k/4$ elements. Applying this logic to the first pivot, where $k = X_0 = n$, we have

$$\begin{aligned}\mathbf{E}[X_1] &= \frac{1}{2}\,\mathbf{E}[X_1 \mid \text{first pivot is good}] + \frac{1}{2}\,\mathbf{E}[X_1 \mid \text{first pivot is bad}] \\ &\leq \frac{1}{2} \cdot \frac{3}{4}n + \frac{1}{2}n \leq \frac{7}{8}n.\end{aligned}$$

More generally, for each index i, *conditional on* X_{i-1}, we have

$$\mathbf{E}[X_i \mid X_{i-1}] \leq \frac{1}{2} \cdot \frac{3}{4}X_{i-1} + \frac{1}{2}X_{i-1} = \frac{7}{8}X_{i-1}.$$

We now claim, by induction on i, that $\mathbf{E}[X_i] \leq (7/8)^i n$. The base case $i = 0$ is immediate. For $i > 0$, we have

$$\mathop{\mathbf{E}}_{X_i}[X_i] \overset{\text{(b)}}{=} \mathop{\mathbf{E}}_{X_{i-1}}\left[\mathop{\mathbf{E}}_{X_i}[X_i \mid X_{i-1}]\right] = \mathop{\mathbf{E}}_{X_{i-1}}\left[\frac{7}{8}X_{i-1}\right] \overset{\text{(c)}}{=} \left(\frac{7}{8}\right)^i n.$$

Here, in (b), we applied the law of iterated expectations. (c) is by our induction hypothesis.

Now, let $k = 32 \ln n$. We have

$$\mathbf{E}[X_k] \leq \left(1 - \frac{1}{8}\right)^k n \overset{\text{(d)}}{\leq} e^{-k/8}n = e^{-4\ln n}n = \frac{1}{n^3}.$$

Here, in (d), we applied the inequality $1 + x \leq e^x$ which holds for all x. Finally, by Markov's inequality, we have

$$\mathbf{P}[D_e \geq k] \leq \mathbf{P}[X_k \geq 1] \leq \mathbf{E}[X_k] \leq 1/n^3,$$

as claimed in Eq. (1.1). This completes the proof. □

1.4 Additional notes and materials

Quicksort is also covered in [MR95, Chapter 1]. This particular proof of the high probability bound for `quick-sort` is from [Har19]. It is implicitly similar to a more standard proof using *concentration inequalities*, which will be introduced later.

Lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2025 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2023 lecture materials. Click on the links below for the following files:

- Handwritten .pdf prepared before the lecture (and .note file).
- Handwritten .pdf annotated during the presentation (and .note file).
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2022 lecture materials. We covered the randomized algorithm for 3-SAT (and not derandomization) and `quick-select` (expected running time only), as well as a randomized algorithm for minimum cut which will be discussed later. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

1.5 Exercises

Additional exercises may be found it [MR95, Chapter 1].

Exercise 1.1. This exercise is about how for many intents and purposes, we approximately have the extremely convenient identity, "$1 + x \approx e^x$".

1. Prove that for all $x \in \mathbb{R}$, $1 + x \leq e^x$.

   Hint: At $x = 0$, both sides are equal. What are their respective rates of change moving away from 0?

2. Prove that for all $x \leq 1$, $e^x \leq 1 + x + x^2$.

Exercise 1.2. Recall the `quick-select` algorithm introduced in Section 1.1.3. The goal of this exercise is to prove that `quick-select` takes $O(n)$ time in expectation. Below we present two different approaches which offer two different perspectives. Both analyses should use linearity of expectation and we ask you to point this out explicitly.

1. *Approach 1.* Analyze `quick-select` similarly to `quick-sort`, based on the sum of indicators X_{ij}.

   One approach is to reduce to a separate analysis for each of the following 4 classes of pairs:

   (a) X_{ij} where $i < j < k$,
   (b) X_{ij} where $i < k < j$,
   (c) X_{ij} where $k < i < j$, and
   (d) X_{ij} where either $i = k$ or $j = k$.

   For each case, show that the expected sum is $O(n)$.

2. *Approach 2.* The following approach can be interpreted as a randomized divide and conquer argument. We are arguing that with constant probability, we decrease the input by a constant factor, from which the fast (expected) running time follows.

   (a) Consider again `quick-select`. Consider a single iteration where we pick a pivot uniformly at random and throw out some elements. Prove that with some constant probability p, we either sample the kth element or throw out at least $1/4$ of the remaining elements.

   (b) For each integer i, prove that the expected number of iterations (i.e., rounds of choosing a pivot) of `quick-select`, where the number of elements remaining is in the range $[(4/3)^i, (4/3)^{i+1})$, is $O(1)$.[3]

[3] Hint: Exercise C.35.

(c) Fix an integer i, and consider the amount of time spent by `quick-select` while the number of elements remaining is greater than $(4/3)^{i-1}$ and at most $(4/3)^{i}$. Show that that the expected amount of time is $\leq O((4/3)^{i})$

(d) Finally, use the preceding part to show that the expected running time of `quick-select` is $O(n)$.

Exercise 1.3. Last week you decided to clean up your toolshed, but you compulsively made the following silly mistake. Originally you had n matching pairs of nuts and bolts of different sizes, but you decided to organize the nuts and bolts separately into a box full of nuts and a box full of bolts, splitting up all the pairs of nuts and bolts in the process. So now you have n nuts, and n bolts, such that each nut fits a distinct bolt and vice-versa, but you don't know which nut goes with which bolt.

The bolts all look pretty similar, so you can't compare two bolts directly with each other and tell which one is bigger. Likewise you cannot compare the nuts to each other. However, you can try to fit one nut to one bolt, and see if either:

1. The nut fits the bolt.
2. The nut is too big for the bolt.
3. The nut is too small for the bolt.

We will treat one of these nut-to-bolt tests as a single operation. Your goal is to match up all nuts and bolts. Of course you can compare every pair of nut and bolt in $O(n^2)$ time, but can you do better?

Design and analyze an algorithm, as fast as possible, to recover all matching pairs of nuts and bolts.

Exercise 1.4. This exercise is about a simple randomized algorithm for *verifying* matrix multiplication. Suppose we have three $n \times n$ matrices A, B, C. We want to verify if $AB = C$. Of course one could compute the product AB and compare it entrywise to C. But multiplying matrices is slow: the straightforward approach takes $O(n^3)$ time and there are (more theoretical) algorithms with running time roughly $O(n^{2.37\ldots})$. We want to test if $AB = C$ in closer to n^2 time.

The algorithm we analyze is very simple. Select a point $x \in \{0,1\}^n$ uniformly at random. (That is, each $x_i \in \{0,1\}$ is an independently sampled bit.) Compute $A(Bx)$ and Cx, and compare their entries. (Note that it is much faster to compute

$A(Bx)$ than AB.) If they are unequal, then certainly $AB \neq C$ and we output `false`. Otherwise we output `true`. Note that the algorithm is always correct if $AB = C$, but could be wrong when $AB \neq C$. We will show that if $AB \neq C$, the algorithm is correct with probability at last $1/2$.

1. Let $y \in \mathbb{R}^n$ be a fixed nonzero vector, and let $x \in \{0,1\}^n$ be drawn uniformly at random. Show that $\langle x, y \rangle \overset{\text{def}}{=} \sum_{i=1}^{n} x_i y_i \neq 0$ with probability at least $1/2$. [4]
2. Use the preceding result to show that if $AB \neq C$, then with probability at least $1/2$, $ABx \neq Cx$.[5]
3. Suppose we want to decrease our probability of error to (say) $1/n^2$. Based on the algorithm above, design and analyze a fast randomized algorithm with the following guarantees.
   - If $AB = C$, then it always reports that $AB = C$.
   - If $AB \neq C$, then with probability of error at most $1/n^2$, it reports that $AB \neq C$.

   (Your analysis should include the running time as well.)

Exercise 1.5. Let P be a set of n distinct points in the plane. Consider the problem of computing the minimum distance $\|p - q\|$ between any two distinct points $p, q \in P$. (If $|P| \leq 1$, the minimum distance is defined as $+\infty$.) For simplicity we assume the points are in general position, so that all pairwise distances are unique. You may have learned a deterministic divide-and-conquer algorithm that runs in $O(n \log n)$ time. Here we will analyze the running time of a different randomized algorithm. To help build intuition, first solve the following problem.

1. (3 points) Let $a_1, \ldots, a_n \in \mathbb{R}$ be n distinct numbers presented in a uniformly random order. Imagine tracking the minimum as you examine the numbers one by one in (the randomized) order. What is the expected number of times the minimum changes?

[4]*Hint: Suppose for simplicity that the last coordinate of y is nonzero. It might help to imagine sampling the first $n-1$ bits and computing the partial sum $S_{n-1} = \sum_{i=1}^{n-1} x_i y_i$ first, before sampling x_n and adding $x_n y_n$. Formally your analysis may involve some conditional probabilities. (And what about the case where $y_n = 0$?)*

[5]Even if you haven't solved part 1 you may assume it to be true.

We assume black box access to a data structure for the following "threshold" version of the minimum distance problem. Fix a threshold $\rho > 0$. The data structure takes $O(1)$ time to initialize, and starts with an empty point set. It supports the following operation:

> `insert(`x`)`: Insert a point $x \in \mathbb{R}^2$ into the underlying point set. If the minimum distance in the underlying point set is (strictly) less than ρ, then return `true`; otherwise return `false`. This operation takes $O(1)$ expected time.[6]

This subroutine, combined with part 1, suggests the following algorithm. Here we assume that one can do math operations (add, subtract, square, etc.) in $O(1)$ time, and that we can sample a random permutation of n items in $O(n)$ time.

```
1. Let $p_1, p_2, \ldots, p_n$ permute $P$ uniformly at random.        // $O(n)$ time.
2. Let $r_1 = +\infty$ and $r_2 = \|p_1 - p_2\|$.
/* We maintain the invariant that $r_i$ is the minimum distance over $p_1, \ldots, p_i$. */
3. Start a new instance of the threshold data structure with threshold $r_2$. Insert
   $p_1$ and $p_2$.
4. For $i = 3, \ldots, n$:
   A. Insert $p_i$ into the threshold data structure. If the data structure returns
      true:
      1. Let $r_i = \min\{\|p_i - p_j\| : j < i\}$.
      2. Replace the threshold data structure with a new one with radius $r_i$.
         Insert $p_1, \ldots, p_i$ into the data structure.
   B. Otherwise set $r_i = r_{i-1}$.
5. Return $r_n$.
```

Now analyze this algorithm via the following steps.

2. (3 points) For each index $i > 2$, analyze the exact probability that $r_i < r_{i-1}$.

[6]We can implement this data structure by building a "hash table over grid cells". Initially we have an empty dictionary for an empty point set. Given a point $x = (x_1, x_2)$, compute the key `key(`x`)` $= (\lfloor x_1/(\rho/\sqrt{2})\rfloor, \lfloor x_2/(\rho/\sqrt{2})\rfloor$ in $O(1)$ time. (Rounding is not a standard operation in all models of computation, but here we assume it takes $O(1)$ time.) If this key is already occupied by a point $y \in P$, then $\|x - y\| \leq \rho$, and we return `true`. Otherwise consider the 8 "neighboring" keys/cells where we add or subtract at most one or zero to each of the key's coordinates. Assuming the minimum distance was $\geq \rho$ before inserting x, each of these 8 cells can have at most 1 point. If the distance between x and any of the $O(1)$ points in neighboring cells is $< \rho$ from x, return `true`.

3. (4 points) Finally, bound the expected running time of the algorithm.

Exercise 1.6. You have a sequence of n switches $S_1, \dots, S_n$ that jointly control m light bulbs $L_1, \dots, L_m$. Each switch can be "up" or "down", and this controls whether the light bulbs are on or off.

Each light bulb L_i, is associated with two sets of switches $A_i, B_i \subseteq [n]$. The switches in A_i turn on the light bulb when they are "up" and the switches in B_i turn on the light bulb then they are "down".

More precisely, for each $j \in A_i$, having switch S_j "up" automatically turns on the light bulb. (It only takes one of these switches to be "up" to turn on the light bulb.) For each $j \in B_j$, turning the switch "down" automatically turns on the light bulb. (Again, it only takes one of these switches to be "down" to turn on the light bulb.)

Thus, for a light bulb L_i, the light bulb L_i lights up if and only if either (a) some switch in A_i is flipped up or (b) some switch in B_i is flipped down. A_i and B_i are generic subsets of switches, not necessarily disjoint, and their union does not necessarily include all the switches. We do assume, however, that $|A_i| + |B_i| \geq 2$ for all i. We assume that the sets A_i and B_i are given explicitly for each i (for simplicity; otherwise they can be obtained by inspection).

Your algorithm can flip switches "up" and "down". For the sake of running times, assume that flipping a single switch takes $O(1)$ time, and inspecting whether a single light bulb is on or off takes $O(1)$ time. The light bulbs turn on and off instantly when you flip a switch.

For each of the following decision problems, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

1. Decide if there exists a way to flip the switches to turn on all the light bulbs.
2. Decide if there exists a way to flip the switches to turn on at least three-fourths of the light bulbs.

Exercise 1.7. Let $G = (V, E)$ be an undirected graph with m edges and n vertices, vertex weights $w : V \to \mathbb{R}_{>0}$, and maximum (unweighted) vertex degree Δ. We let $W = \sum_{v \in V} w(v)$ denote the total weight of all the vertices.

Recall than an *independent set* is a set of vertices $S \subseteq V$ such that no two vertices in S are connected by an edge. Computing the maximum cardinality independent set is NP-Hard. In fact, for all $\epsilon > 0$, it is NP-Hard to get a $1/n^{1-\epsilon}$-approximation to the maximum cardinality independent set. (Of course, computing the maximum *weight* independent set is no easier.)

Consider the following algorithm that always returns a *maximal* independent set:

```
randomized-greedy($G = (V, E)$, $w$)
1. Order the vertices $v_1, \dots, v_n$ uniformly at random.
2. $S \leftarrow \emptyset$.
3. For $i = 1, \dots, n$:
   A. If $S + v_i$ is independent, then set $S \leftarrow S + v_i$.
4. Return $S$.
```

1. (6 points) Analyze the approximation ratio of `randomized-greedy`, as a function of the maximum degree Δ.
2. (4 points) Derandomize `randomized-greedy`. That is, design and analyze a deterministic polynomial-time algorithm that outputs an independent set with weight at least as good as the expected weight of the independent set returned by `randomized-greedy`.

Exercise 1.8. Prove Turán's theorem:

> **Turan's theorem***Any undirected graph G with m edges and n vertices has an independent set of size (at least)*
>
> $$\frac{n^2}{2m + n}.$$

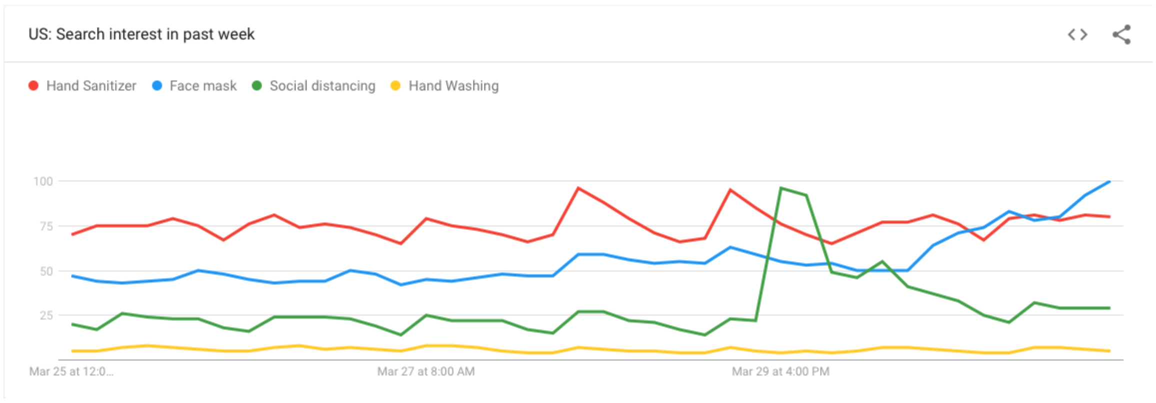
(You may want to solve Exercise C.70 first.)

Chapter 2

Hashing and heavy hitters

2.1 Introduction

Google has an interesting web page called "Google trends", which tracks surging search queries around the world in real time. In Spring 2021, there was even a subpage of search trends specifically related to the Covid-19.



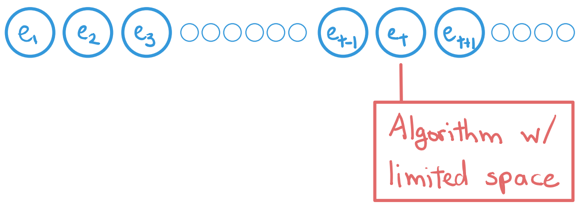
Google tracks the trending search queries not just for the sake of curiosity. Its goal is not only to serve queries, but to serve queries *fast*. The best way to serve something quickly is to have it ready before it is even asked for. By keeping track of the 'heavy hitter" search terms - a few search terms that make up a disproportionate amount of the search traffic - Google can cache the answers to most search requests before they are even made.

Google currently serves billions[1] of queries a day. Given the sheer magnitude of Google's search traffic, and the diversity of search queries, it is not obvious how to

[1]Maybe 7 billion? See https://www.internetlivestats.com/google-search-statistics/

identify the most popular search queries. Certainly one cannot simply have a tally for each search term, since there are too many search terms out there to be stored. More generally, it is prohibitively expensive to maintain any data structure proportional to the input size. Somehow we need an approach that takes *sublinear* space.

Streaming models. We study the heavy hitter problem in the *streaming model* of computation. In the streaming model, the input is a sequence of items presented to the algorithm *one at a time*. We assume the algorithm has much less memory than the size of the input. In particular, it cannot simply write down everything and solve the problem offline. For example, suppose the stream had m items in the stream, and the algorithm was allowed only $O(\sqrt{m})$ space. When the space is so much smaller than the input size, each time an item from the stream is given to the algorithm, the algorithm needs to fairly selective about what parts of the item (if any) it wants to remember.



The heavy hitters problem. We formalize the heavy hitters problem as follows. We have m elements in a stream $e_1, \ldots, e_m$, where each element is from some large universe $[n] = \{1, \ldots, n\}$. Elements can repeat. The *absolute frequency* of an element e, denoted f_e, is the number of times the element appears in the stream. The *relative frequency*, denoted p_e, is the fraction of the stream that the element appears in. In a stream of m elements, the relative frequency of an element is the absolute frequency divided by m.

Ideally we would track all of the elements exactly. This is impossible with less than $\min\{n, m\}$ bits[2]. Here, and unlike standard algorithmic settings, upper bounds like $O(m)$ or $O(n)$ bits are not good enough. So we relax our requirements and formulate problems that are more *tractable* but still *useful*. We first relinquish exactness and consider *approximations*, where we allow for some error that we can analyze and control. We also give up on deterministic computation, and design *randomized* algorithms that have some small — analyzed and controlled — *probability of failure*.

[2]One can formalize this impossibility as follows. There are $\binom{m+n-1}{n-1}$ ways to make a frequency vector of n items where the total sum is m. (*Why?*) Suppose we claim that we can describe any combination of counts with k bits. Each k-bit string can describe by only one of these $\binom{m+n-1}{n-1}$

ϵ-heavy hitters. Given a fixed parameter $\epsilon \in (0,1)$, an *ϵ-heavy hitter* is an element with relative frequency $\geq \epsilon$. The heavy hitters problem is to identify all of the ϵ-heavy hitters for an input parameter $\epsilon > 0$. Note that there can only be $(1/\epsilon)$-many ϵ-heavy hitters, which preserves some hope that we can identify all of them with space proportional to $1/\epsilon$, rather than m.

We also consider a closely related problem of *(approximate) frequency estimation*. Given a fixed error parameter $\epsilon > 0$, the goal is to estimate every element's relative frequency up to an additive error of ϵ. Equivalently, we want to estimate the absolute frequency of each element up to an ϵm-additive factor. At first it might seem impossible to estimate n-many counts with $o(n)$ space. We have already argued that m exact counters is impossible. Allowing for ϵ-relative error, however, means that 0 is a satisfactory estimate for all but at most $\lfloor 1/\epsilon \rfloor$ elements.

If we had a data structure that can estimate the absolute frequency of each element up to an additive error parameter ϵ, then one can use such a data structure with (approximately) maintain a list of all the heavy hitters. See exercise 2.3. Of course frequency estimation gives more information than just who are the heavy hitters. By knowing their frequencies up to some small error, one can also rank them (approximately) from most to least frequent, such as in Google trends.

Conversely, if we knew *a priori* which of the elements are the ϵ-heavy hitters, then ϵ-frequency estimation is trivial. Namely, we would maintain a counter for each of the $\lfloor 1/\epsilon \rfloor$-many ϵ-heavy hitters. All other elements are ignored and assigned frequency 0. Of course this approach is not possible since we do not know the heavy hitters.

Surprisingly we will pursue a strategy that is actually quite similar. We will allocate $w = O(1/\epsilon)$ counters, hoping to use one counter for each heavy hitter (plus a few extra for safe measure). Although we do not know the heavy hitters, we will use *randomized hash functions* to isolate them implicitly.

---

outcomes, so we must have $2^k \geq \binom{m+n-1}{n-1}$, hence $k \geq \log\left(\binom{m+n-1}{n-1}\right)$. Here log denotes $\log_2$. We also have

$$\binom{n+m-1}{n-1} = \left(\frac{n+m-1}{n-1}\right)^{n-1} \text{ and also } \binom{n+m-1}{n-1} = \binom{n+m-1}{m} \geq \left(\frac{n+m-1}{m}\right)^m$$

Thus

$$k \geq (n-1)\log\left(1 + \frac{m}{n-1}\right) \text{ and } k \geq m \log\left(1 + \frac{n-1}{m}\right).$$

Slightly better low bounds can be obtained via *Stirling's approximation*, which is also related to entropy.

2.2 Hashing

Likely the reader has used hash tables before, and may be aware that they use *hash functions* to randomly map keys to slots in an array. (We discuss hash tables in detail in the following chapter.)

Loosely speaking, a hash function is a randomly constructed function $h : [n] \to [k]$ where the values $h(i)$ are (in a qualified sense) randomly distributed through $[k]$. A *collision* is a pair of distinct indices $i_1 \neq i_2 \in [n]$ such that $h(i_1) = h(i_2)$. In most applications, n is much (much, much) larger than k. In this case, there are necessarily many collisions (see exercise C.6). A goal of hash functions is to distribute these collisions "fairly".

Ideal hash functions. One way to construct a hash function, for example, is to sample, for each $i \in [n]$, a value $h(i) \in [k]$ independently and uniformly at random. This produces an "ideal hash function", defined as follows.

Definition 2.1. *An* ideal hash function $h : [n] \to [k]$ *is a uniformly random function* $h : [n] \to [k]$*. That is, each* $h(i)$ *is drawn from* $[k]$ *independently and uniformly at random.*

An ideal hash function h is particularly easy to reason about. For example, for every input i and possible output $j \in [k]$, we have

$$\mathbf{P}[h(i) = j] = \frac{1}{k}.$$

More generally, for any ℓ distinct inputs $i_1, \dots, i_\ell \in [n]$ and ℓ possible outputs $j_1, \dots, j_\ell \in [\ell]$, we have

$$\begin{aligned} &\mathbf{P}[h(i_1) = j_1,\, h(i_2) = j_2,\, \dots,\, h(i_\ell) = j_\ell] \\ &= \mathbf{P}[h(i_1) = j_1]\,\mathbf{P}[h(i_2) = j_2] \cdots \mathbf{P}[h(i_\ell) = j_\ell] \\ &= \frac{1}{k^\ell}. \end{aligned}$$

Ideal hash functions are a good model to keep in mind when designing randomized algorithms. Assuming the hash values are completely independent simplifies calculations. In reality, however, ideal hash functions are very expensive to make and store. Indeed, one has to have $n \log k$ bits to be able to describe all of the possible functions from $[n]$ to $[k]$ (as there are k^n such functions, and we must pay the logarithm of this quantity). This is particularly ill-suited to our streaming setting where n is astronomical and the goal is to use space sublinear to the input size.

Universal hash functions. Fortunately, in most applications, only a *limited amount of randomization* is actually required. For the current discussion, we only require "universal" hash functions that have "ideal pairwise collision probabilities", in the following sense.

Definition 2.2. *A random hash function $h : [n] \to [k]$ is* universal *if for any distinct indices $i_1 \neq i_2 \in [n]$, we have*

$$\mathbf{P}[h(i_1) = h(i_2)] \leq \frac{1}{k}.$$

In contrast to ideal hash functions, universals hash functions can be constructed compactly, as described in the following theorem.

Theorem 2.3. *Consider the randomly constructed function $h : [n] \to [k]$*

$$h(x) = (ax + b \mod p) \mod k,$$

where p is a prime number larger than n, $a \in \{1, \dots, p-1\}$ is drawn uniformly at random, and $b \in \{0, \dots, p\}$ is drawn uniformly at random. Then h is a universal hash function.

The proof of Theorem 2.3 is given as exercise 2.2. Here we will assume Theorem 2.3 and focus on its application to the heavy hitters problem.

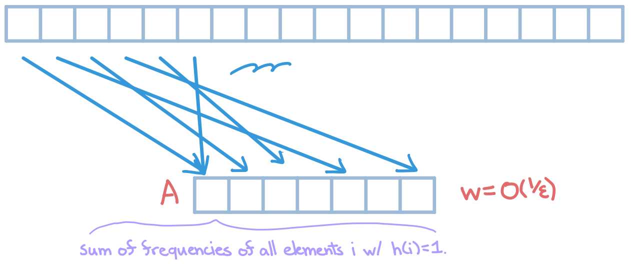
2.3 Using hashing to approximate frequencies

Let us now return to the frequency estimation problem. We have elements from the set $[n]$ arriving in a stream. We assume we know n *a priori* but not the length of the stream, which might as well be infinite. In the analysis, we imagine pausing the stream at a fixed point in time after m elements have arrived, and analyze the algorithm at that point in time.

The goal is to estimate the absolute frequency of each element up to an (ϵm)-additive factor. The crux of the problem is that total space usage should be (more or less) independent of the length of the stream, m, or the number of elements, n. We mentioned briefly above that if we knew the heavy hitters, then we could just maintain a counter for each one. Since there are at most $1/\epsilon$ heavy hitters, this approach would satisfy our space constraints. Of course we do not know the heavy hitters. In the following, we will use hash functions to, in effect, *guess* the heavy hitters.

```
hashed-counters($\epsilon > 0$)
1. Allocate an array of size $A[1..w]$ for $w = \lceil 2/\epsilon \rceil$
2. Sample a universal hash function $h : \{1, \dots, n\} \to \{1, \dots, w\}$
3. For each item $e$ in the stream
    A. $A[h(e)] \leftarrow A[h(e)] + 1$
```

Figure 2.1: Hashing into a $O(1/\epsilon)$ counters.



We first create an array of counters $A[1..w]$ with $w = \lceil 2/\epsilon \rceil$ entries. Note that $2/\epsilon$ is extremely small compared to the total length of the stream, or the distinct number of keys. We also sample a universal hash function $h : \{1, \dots, n\} \to \{1, \dots, w\}$. For each element e presented by the stream, we increase $A[h(e)]$ by 1. In turn, for each element e, we treat $A[h(e)]$ as an estimate for f_e. See Fig. 2.1 for pseudocode.

$A[h(e)]$ never underestimates f_e, and the hope is that it does not overestimate f_e by too much. The risk of error comes from other elements' frequencies possibly adding more than ϵm to $A[h(e)]$. Here the intuition is that the "noise" coming from other frequencies is spread out by the hash function over $2/\epsilon$ entries, so we would only expect $\epsilon m/2$ error for each element e. To translate "expected error" to "probability of error", we use Markov's inequality, as follows.

Lemma 2.4. *For each element e, with probability $\geq 1/2$, we have*

$$f_e \leq A[h(e)] \leq f_e + \epsilon m.$$

Proof. We have $A[h(e)] \geq f_e$ always because $A[h(e)]$ is a sum of frequencies of elements with hash code $h(e)$, which of course includes e. The expected additive error is bounded above by

$$\mathbf{E}[A[h(e)]] - f_e \overset{\text{(a)}}{=} \sum_{d \neq e} f_d \, \mathbf{P}[h(d) = h(e)] \overset{\text{(b)}}{\leq} m/w \leq \frac{\epsilon}{2} m. \tag{2.1}$$

Here (a) is by linearity of expectation. (b) is because h is universal. Now we have

$$\mathbf{P}[A[h(e)] \geq f_e + \epsilon m] \overset{\text{(c)}}{\leq} \mathbf{P}[A[h(e)] - f_e \geq 2\,\mathbf{E}[A[h(e)] - f_e]] \overset{\text{(d)}}{\leq} \frac{1}{2}$$

```
count-min($\epsilon > 0$, $\delta > 0$)
1. build $d = \lceil \log 1/\delta \rceil$ instances $(A_1, h_1), \dots, (A_d, h_d)$ of hashed-counters($\epsilon$)
   over the stream
2. to query an element $e$:
   A. return $\min_{i=1,\dots,d} A_i[h_i(e_i)]$
```

Figure 2.2: The `count-min` data structure

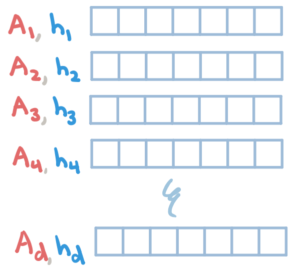
Here (c) plugs in the inequality obtained in (2.1). (d) applies Markov's inequality, where we note that $A[h(e)] - f_e \geq 0$. □

Given that it is impossible to track frequencies exactly in sublinear space, it is suprising that we can now count every element's frequency with extremely small space, *sometimes and with small additive error.* The algorithm, including the construction of the universal hash function, is extremely simple.

2.4 Amplification

Section 2.3 shows how to estimate each element with fairly small error with constant probability of error. Our goal now is to reduce the error probability enough to even take the union bound over all of the elements, and thus estimate all frequencies up to ϵm-additive error.

The idea is to use repetition, and one analogy is coin tossing. The goal is to flip enough coin tosses to get at least one heads with very high probability. With one coin toss, the probability that it is tails is $1/2 = .5$. With two coin tosses, the probability that both come up tails is still $1/4 = .25$. But with 100 coin tosses, the probability that all 100 coin tosses come up tails is $1/2^{100} \approx .0000000000000000000000000000007886\dots$.



The point is that independent trials magnify the probability of success exponentially. For a specified probability of error $\delta \in (0,1)$, the algorithm `count-min` below makes $\lceil \log 1/\delta \rceil$ independent instances of `hashed-counters(`ϵ`)`. For each element e, it uses the minimum estimate over all of the instances of `hashed-counters`. The overall data structure fails for an element e only if *every* instance of `hashed-counters` fails, which by the analogy with coins, is exceedingly unlikely.

We now have the following smaller error probability for each element e.

Lemma 2.5. *For each element e, with probability $\geq 1 - \delta$, we have*

$$\min_{i=1,\dots,d} A_i[h_i(e)] \leq f_e + \epsilon m.$$

Proof. We have

$$\mathbf{P}\left[\min_{i=1,\dots,d} A_i[h_i(e)] > f_e + \epsilon m\right] \overset{\text{(a)}}{=} \prod_{i=1}^{d} \mathbf{P}[A_i[h_i(e)] > f_e + \epsilon m] \overset{\text{(b)}}{\leq} \frac{1}{2^d} \leq \delta.$$

Here (a) is by independence of each $A_i[h_i(e)]$. (b) is by Lemma 2.4. □

For δ set to a polynomial of $1/n$, the probability of error becomes low enough to take a union bound over all elements in $[n]$, as follows.

Theorem 2.6. *Given a stream of elements from the range $[n]$,* `count-min(`ϵ`,`$1/n^2$`)` *has the following guarantee at any fixed point in the stream.*

Suppose m elements have been presented in the stream. With probability at least $1 - 1/n$, `count-min(`ϵ`,`$1/n^2$`)` *overestimates the total frequency of each element with additive error at most ϵm and total space $O(\log(n)/\epsilon)$.*

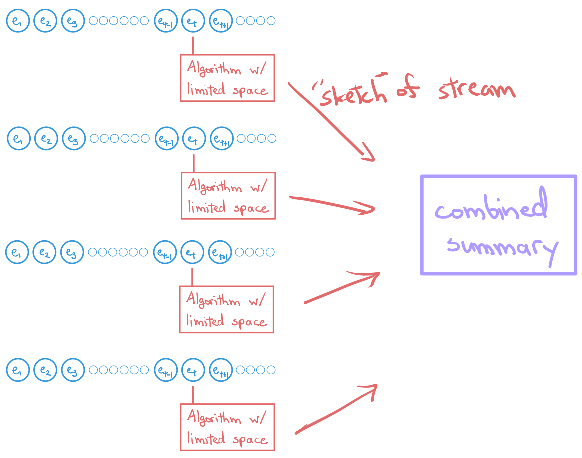
Proof. By Lemma 2.5, we have probability of error $\leq 1/n^2$ for each element e. Taking the union bound over all n elements in the stream, we have probability of error $\leq 1/n$. □

Remark 2.7. More precisely, the space usage of `count-min(`ϵ`,`$1/n^2$`)` is that of $O(\log(n)/\epsilon)$ counters. Here we assume each counter takes $O(1)$ space for simplicity.

2.5 Extensions

2.5.1 Crossing streams

One can extend the streaming model to multiple streams in the following *distributed* model of computation. Here we have several streams simultaneously, each served by an algorithm using sublinear space. The goal is to solve the heavy hitters problem over the *combined streams*.



`count-min` has the convenient property of being a *sketch*. To handle multiple streams, we have an instance of `count-min` for each stream arranged so that they are *all using the same hash functions*. To combine their results, we simply sum up the arrays A_i of hashed sums entry-wise. The result is an instance of `count-min` over the combined streams.

2.5.2 Turnstile streams

Consider the more general model where each item in the stream consists of an element e and a value Δ, signifying that we should increase the frequency count for e, f_e by Δ. Δ is allowed to be negative, with the restriction that the frequency f_e of each element (which is now the sum of $\Delta's$ for that element) remains nonnegative. This model is sometimes called a "turnstile stream", in the sense that a turnstile counting the number of people in an amusement park is always nonnegative because each decrease corresponds to a person who entered the park earlier.

`count-min` adapts immediately to turnstile stream, by simply adding Δ to $A_i[h_i(e)]$ for each instance (A_i, h_i) of `hashed-counters`. The additive error is now ϵ times the sum of all Δ's in the stream.

2.6 Takeaways

- There are many basic and useful problems – heavy hitters with sublinear space being on of our first examples – that are too difficult or even impossible to compute exactly and deterministically. Instead we consider *randomized approximation algorithms* that are potentially more scalable. This requires *quantitative analysis* to address the approximation factor in addition to algorithm design.

- `count-min-sketch` uses *hashing* to try to distribute the heavy hitters across an array. It does not know which are the heavy hitters, but relies on *randomization* to separate the heavy hitters (most of the time) in an *oblivious* fashion.
- *Ideal hash functions*, while easy to reason about, are prohibitively expense. Luckily, weaker hash functions with *limited randomness* often suffice, and are easily constructed. `count-min-sketch` requires only *universal hash functions*. Universal hash functions can be implemented very easily.
- *Linearity of expectation*, combined with universal hash functions, implies that the noise seen by a particular element is evenly spread out *on average*. *Markov's inequality* allowed us to argue that the noise encountered by an element is close to the average, most of the time.
- `count-min` *amplifies* the probability of success by taking the minimum over many independent trials. A particular element is miscounted if and only if *all* independently trials miscount the element, which happens with vanishingly small probability.
- The error probability drops so rapidly that we can apply the *union bound* over all of the elements after just $O(\log n)$ trials.
- `count-min` does not give *unbiased estimates* of the counters. Instead, `count-min` tries to be *within a prescribed error with high probability*. It is *consistent*. It is more important to be *consistent* then *unbiased*, since we can (psychologically) adjust for the bias. Many real-world apparatus are designed on this principle.

2.7 Additional notes and references

The `count-min` data structure is from [CM05]. Additional notes can be found in [Nel20; Che14].

Lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2025 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2023 lecture materials. Click on the links below for the following files:
- Handwritten .pdf prepared before the lecture (and .note file).
- Handwritten .pdf annotated during the presentation (and .note file).
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

2.8 Exercises

Exercise 2.1. Let $h : [n] \to [k]$ be *any* fixed function.

1. Prove that the number of collisions is

$$\geq \frac{n(n-k)}{2k}$$

2. Show that the above inequality is tight when k divides n.

Exercise 2.2. Show that the construction given in Section 2.2 is indeed a universal hash function, using the steps listed below.

To recall the construction, we randomly construct a function $h : [n] \to [k]$ as follows. First, let p be any prime number $> n$. Draw $a \in \{1, \dots, p-1\}$ uniformly at random, and draw $b \in \{0, \dots, p-1\}$ uniformly at random. We define a function $h(x)$ by

$$h(x) = ((ax + b) \mod p) \mod k.$$

1. Let $x_1, x_2 \in [n]$ with $x_1 \neq x_2$, and let $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$. Show that the system of equations

$$\begin{aligned} ax_1 + b &= c_1 \mod p \\ ax_2 + b &= c_2 \mod p \end{aligned}$$

uniquely determines $a \in \{1, \dots, p-1\}$ and $b \in \{0, \dots, p-1\}$.[3]

- Step 1 implies that the map $(a, b) \mapsto (ax_1 + b \mod p, ax_2 + b \mod p)$ is a bijection between $\{1, \dots, p-1\} \times \{0, \dots, p-1\}$ and $\{(c_1, c_2) \in \{0, \dots, p-1\} : c_1 \neq c_2\}$.

2. Let $x_1, x_2 \in [n]$ with $x_1 \neq x_2$, and let $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$. Show that

$$\mathbf{P}[ax_1 + b = c_1 \mod p,\ ax_2 + b = c_2 \mod p] = \frac{1}{p(p-1)}.$$

(Here the randomness is over the uniformly random choices of a and b.)

3. Fix $x_1, x_2 \in [n]$ with $x_1 \neq x_2$, and $c_1 \in \{0, \dots, p-1\}$. Show that

$$\sum_{\substack{c_2 \in \{1, \dots, p\} \\ c_2 \neq c_1 \mod p \\ c_1 = c_2 \mod k}} \mathbf{P}[ax_1 + b = c_1 \mod p,\ ax_2 + b = c_2 \mod p] \leq \frac{1}{pk}.$$

- The LHS represents $\mathbf{P}[ax_1 + b = c_1 \mod p \text{ and } h(x_2) = h(x_1)]$.[4][5]

[3] Here it is helpful to know that division is well-defined on the set of integers modullo p when p is prime. More precisely, "a/b" is defined as the unique integer c such that $bc = a$.

[4] Here we note that for $x_1 \neq x_2$, $ax_1 + b = ax_2 + b_2 \mod p$ iff $a = 0$.

[5] *Hint:* You may want to show that the number of values $c_2 \in [p]$ such that $c_1 = c_2 \mod k$ is $\leq \frac{p-1}{n}$.

4. Finally, show that $\mathbf{P}[h(x_1) = h(x_2)] \leq \frac{1}{k}$.

Exercise 2.3. The `count-min` data structure allows us to estimate the relative frequency of each element up to an ϵ-additive factor with probability of error $\leq 1/\operatorname{poly}(n)$ with $O(\log(n)/\epsilon)$ space. [6] The original motivation, however, was to also obtain a list of ϵ-heavy hitters. Design and analyze an algorithm that maintains a list of elements, with at any *particular* point in time,[7] with probability of error $\leq 1/n^2$:

1. Contains all of the ϵ-heavy hitters.

2. Only includes $(\epsilon/2)$-heavy hitters.

Your space usage should be comparable to the space used by the `count-min` data structure.[8]

Additional remark. The question asks for *one data structure* that satisfies *both the criteria* simultaneously. That is, you should maintain a list S that (a) contains all ϵ-heavy hitters, *and* (b) only includes $(\epsilon/2)$-heavy hitters. The tricky part is that `count-min` only approximates the frequencies. You may want to account for the fact that an instance of `count-min(`ϵ`,`δ`)` may overestimate the relative frequency of an element by as much as ϵ, which can make a very infrequent element look like an ϵ-heavy hitter.

Exercise 2.4. In this exercise, we develop a refined analysis that can reduce the additive error substantially in (arguably realistic) settings when the stream is dominated by heavy hitters.

Let S denote the sum of frequency counts of all elements that are *not* ϵ-heavy hitters:

$$S = \sum_{e : p_e < \epsilon} f_e.$$

Note that $S \leq m$, and S might be much less than m when the stream is dominated by heavy hitters.

[6] Here the elements are integers from $[n] = \{1, \dots, n\}$, where n is known, and $\epsilon \in (0, 1)$ is an input parameter.

[7] To clarify, what we mean by "particular point in time" is as follows. You have a data structure that is processing data over time. Suppose we suddenly paused the stream and asked you to report your list of heavy hitters. Your algorithm should succeed then and there with probability of error $\leq 1/n^2$. For this criteria, you do not need to know the length of the stream.

[8] You may want to use the `count-min(`ϵ`,`δ`)` data structure as a black box, but you should be clear about your choice of parameters ϵ and δ.

Show that, by increasing the parameter w (in the `count-min` data structure) by a constant factor, and still using universal hash functions, one can estimate the frequency of every element with additive error at most ϵS with high probability in $O(\log(n)/\epsilon)$ space.[9]

Exercise 2.5. Consider the streaming model where we have elements $e_1, e_2, \ldots$ presented one at a time by a stream. A natural task is to sample a fixed number of elements uniformly at random from the stream. Usually, sampling (say) 1 item from a set of m elements is easy: randomly generate a number k between 1 and m, and return the kth element form your set. Sampling in streaming is trickier because we cannot hold the entire stream in memory, and don't know the length of the stream.

1. Consider the following randomized streaming algorithm that selects one element s from the stream:

```
sample-one

/* $m$ counts the number of elements in the stream so far, and $s$ is
   the “sample” of 1 element from the stream.                        */
1. $m \leftarrow 0$, $s \leftarrow$ nil.
2. For each element $e$ presented by the stream:
   A. $m \leftarrow m + 1$.
   B. With probability $1/m$:
      1. $s \leftarrow e$.
```

   For $i \in \mathbb{N}$, let e_i denote the ith element in the stream. For $m \in \mathbb{N}$ let s_m denote the value of s after the mth iteration. Show that for all i and m,

$$\mathbf{P}[s_m = e_i] = \begin{cases} 0 & \text{if } m < i \\ 1/m & \text{if } m \geq i. \end{cases}$$

   That is, for each m, s_m is a uniformly random element out of $\{e_1, \ldots, e_m\}$.[10]

2. Now let $k \in \mathbb{N}$ be a fixed parameter. (e.g., $k = 3$.) Suppose you want to sample a set of k elements from the stream without replacement. Design and analyze an algorithm generalizing `sample-one` that maintains a sample S of k elements

[9] *Hint*: It might be helpful to think about the special case of $S = 0$.

[10] Fix i. For $m < i$ the probability 0 since e_i hasn't event appeared in the stream. Now, what about $m = i$? What about $m = i + 1$?

drawn uniformly at random from the stream. That is, for $m \geq k$, your algorithm should have a set S of k elements, where any particular set of k elements is equally likely (i.e., with probability $1/\binom{m}{k}$). For $k = 1$, your algorithm should coincide with `sample-one` above.[11]

Exercise 2.6. This exercise gives a different and interesting application of hashing to string matching. In string matching, we have a long text string $T[1..n]$, and a smaller search string $S[1..k]$, and we want to decide if S occurs in T. For simplicity we assume these are bit strings, but it is easy to generalize to larger alphabets.

The naive approach directly compares $S[1..k]$ to each length-k substring $T[i, ..., i + k - 1]$, and takes $O(nk)$ time. A more sophisticated algorithm due to Knuth, Morris, and Pratt compiles S into a deterministice finite automaton A_S of size $O(k)$, and uses the automaton A_S to search T in $O(n)$ time. Here we take a different approach based on randomization.

Now, a k-bit string $x \in \{0,1\}^k$ can be thought of as an integer

$$2^{k-1}x_1 + 2^{k-2}x_2 + \cdots + 2x_{k-1} + x_k$$

between 0 and $2^k - 1$. One might try to compute all the integers for the k-bit substrings $T[1..k], T[2,,k+1], ..., T[n-k+1..n]$ of T and compare each to the integer form of S. But this is really no different than the naive approach of direct comparison, since the integers are k-bits long.

Suppose instead we took these integers modulo a prime p drawn from a range $\{2, \ldots, q\}$, for sufficiently large q. Consider the hash function $h : \{0,1\}^k \to \mathbb{Z}_{\geq 0}$ defined by

$$h(x_{1..k}) = 2^{k-1}x_1 + 2^{k-2}x_2 + \cdots + 2x_{k-1} + x_k \mod p$$

for a randomly selected prime number p. h is a *rolling* hash function: as we "shift" the hash function by 1-bit from (say) bits $1, ..., k$ to bits $2, ..., k+1$, we need only update

$$h(x_{2..k+1}) = 2(h(x_{1..k}) - 2^{k-1}x_1) + x_{k+1} \mod p$$

[11] One way to frame your analysis is as follows. For $m \geq k$, let S_m denote the (randomized) sample S after m iterations. Prove the following statement by induction on $m - k$:

For all $m \geq k$, and all sets $X \subseteq \{e_1, \ldots, e_m\}$ of k elements,

$$\mathbf{P}[S_m = X] = \frac{1}{\binom{m}{k}}.$$

In our argument, you may have two cases depending on whether or not $e_m \in X$.

with a constant number of arithmetic operations, instead of $O(k)$ operations to compute it from scratch.

The goal is to use the rolling hash function to design and analyze a fast randomized algorithm for string matching. The problem of course is collisions between distinct substrings, and the probability of collision depends on the random selection of p. The following facts about prime numbers may be helpful:

- By the prime factorization theorem, every integer can be represented uniquely as a product of prime numbers.
- By the prime number theorem, there are $(1 - o(1))n/\ln n$ primes between 1 and n.
- There is a deterministic polynomial time algorithm for verifying if a number is prime, and a faster randomized algorithm that succeeds with high probability. In particular, there is a deterministic algorithm that runs in $\tilde{O}(k^6)$ time for k-bit integers.

1. Let $x, y \in \{0,1\}^k$ be two distinct k-bit strings interpreted as integers between 0 and $2^k - 1$. Observe that $x = y \mod p$ iff p divides $|x - y|$, which is again an integer between 0 and $2^k - 1$. Prove that there are at most $\log_2(|x - y|)$ distinct prime numbers dividing $|x - y|$.
2. Suppose p is a random prime number from the range $\{2, \dots, q\}$ for some value q. How large does q need to be to guarantee that p does not divide $|x - y|$ with probability (say) at least 1/2?
3. Using the observations from the previous 2 problems, design and analyze a randomized algorithm searching for S in T that runs in $O(n \operatorname{poly}(\log k))$ time (the faster the better). (Your algorithm should always be correct, and take $O(n \operatorname{poly}(\log k))$ time in expectation. You can assume that arithmetic operations modulo p takes $O(\log p)$ time.)[12]

[12]*Hint:* You might first try to get a $O\left(n \log n \log^{O(1)} k\right)$ time algorithm that succeeds with high probability. (This will still get most of the points.) Getting it down to $O\left(n \log^{O(1)} k\right)$ time is a little trickier.

Chapter 3

Hash tables and linear probing

3.1 Dictionaries

It is difficult to imagine programming without *dictionaries* and *maps*, which are data structures with the following two operations.

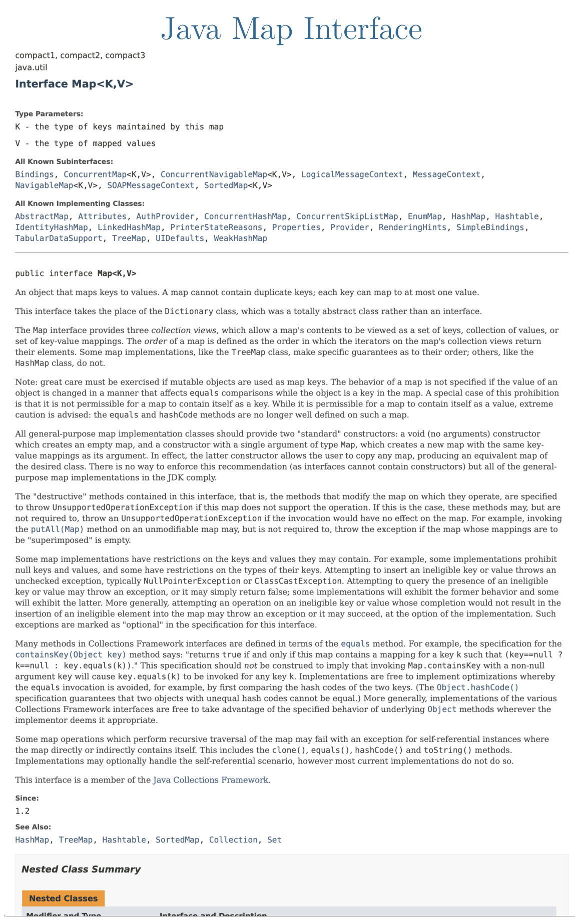
1. `set(`k`,`v`)`: Associate the value v with the key k.
2. `get(`k`)`: Return the value associated with the key k (if any).

These two operations form a dead-simple way to store data that can be used in almost any situation. Inevitably all large software systems, however well-planned and structured and object-oriented initially, end up using and passing around dictionaries to organize most of their data. The embrace of dictionaries is taken to another level in `Python` and `Javascript`. These languages provide dictionaries as a *primitive*, and supply a convenient syntax to make them very easy to use. In fact the class object systems in both of these languages are really just dictionaries initialized by some default keys and values and tagged with some metadata. Screenshots of the online documentation for the `Map` interface in `Java` and for `dict` (short for dictionary) in `Python` are given in Fig. 3.1.

We first point out a special case of the dictionary problem that would be ideal. Suppose that there are n keys, and that they are all integers between 1 and n. Then one can simply allocate an array $A[1..n]$ of size n, to hold the n values. Recall that an array consists of n contiguous slots in memory, and the ith slot, $A[i]$, can be retrieved or rewritten in constant time. There is also a real benefit to the fact that the array physically occupies contiguous spots on the hardware. This physical arrangement implies an extremely compact data structure with fewer cache misses.[1]

While the array is ideal for its particular use case, it is not very flexible either. Adding a new key $k = n + 1$, for example, would require rebuilding a new array of

[1]Sometimes, constant factors matter.



Java Map Interface

compact1, compact2, compact3
java.util

Interface Map<K,V>

Type Parameters:

`K - the type of keys maintained by this map`

`V - the type of mapped values`

All Known Subinterfaces:

`Bindings`, `ConcurrentMap<K,V>`, `ConcurrentNavigableMap<K,V>`, `LogicalMessageContext`, `MessageContext`, `NavigableMap<K,V>`, `SOAPMessageContext`, `SortedMap<K,V>`

All Known Implementing Classes:

`AbstractMap`, `Attributes`, `AuthProvider`, `ConcurrentHashMap`, `ConcurrentSkipListMap`, `EnumMap`, `HashMap`, `Hashtable`, `IdentityHashMap`, `LinkedHashMap`, `PrinterStateReasons`, `Properties`, `Provider`, `RenderingHints`, `SimpleBindings`, `TabularDataSupport`, `TreeMap`, `UIDefaults`, `WeakHashMap`

```
public interface Map<K,V>
```

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

This interface takes the place of the `Dictionary` class, which was a totally abstract class rather than an interface.

The `Map` interface provides three *collection views*, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The *order* of a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the `TreeMap` class, make specific guarantees as to their order; others, like the `HashMap` class, do not.

Note: great care must be exercised if mutable objects are used as map keys. The behavior of a map is not specified if the value of an object is changed in a manner that affects `equals` comparisons while the object is a key in the map. A special case of this prohibition is that it is not permissible for a map to contain itself as a key. While it is permissible for a map to contain itself as a value, extreme caution is advised: the `equals` and `hashCode` methods are no longer well defined on such a map.

All general-purpose map implementation classes should provide two "standard" constructors: a void (no arguments) constructor which creates an empty map, and a constructor with a single argument of type `Map`, which creates a new map with the same key-value mappings as its argument. In effect, the latter constructor allows the user to copy any map, producing an equivalent map of the desired class. There is no way to enforce this recommendation (as interfaces cannot contain constructors) but all of the general-purpose map implementations in the JDK comply.

The "destructive" methods contained in this interface, that is, the methods that modify the map on which they operate, are specified to throw `UnsupportedOperationException` if this map does not support the operation. If this is the case, these methods may, but are not required to, throw an `UnsupportedOperationException` if the invocation would have no effect on the map. For example, invoking the `putAll(Map)` method on an unmodifiable map may, but is not required to, throw the exception if the map whose mappings are to be "superimposed" is empty.

Some map implementations have restrictions on the keys and values they may contain. For example, some implementations prohibit null keys and values, and some have restrictions on the types of their keys. Attempting to insert an ineligible key or value throws an unchecked exception, typically `NullPointerException` or `ClassCastException`. Attempting to query the presence of an ineligible key or value may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible key or value whose completion would not result in the insertion of an ineligible element into the map may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.

Many methods in Collections Framework interfaces are defined in terms of the `equals` method. For example, the specification for the `containsKey(Object key)` method says: "returns `true` if and only if this map contains a mapping for a key `k` such that `(key==null ? k==null : key.equals(k))`." This specification should *not* be construed to imply that invoking `Map.containsKey` with a non-null argument `key` will cause `key.equals(k)` to be invoked for any key `k`. Implementations are free to implement optimizations whereby the `equals` invocation is avoided, for example, by first comparing the hash codes of the two keys. (The `Object.hashCode()` specification guarantees that two objects with unequal hash codes cannot be equal.) More generally, implementations of the various Collections Framework interfaces are free to take advantage of the specified behavior of underlying `Object` methods wherever the implementor deems it appropriate.

Some map operations which perform recursive traversal of the map may fail with an exception for self-referential instances where the map directly or indirectly contains itself. This includes the `clone()`, `equals()`, `hashCode()` and `toString()` methods. Implementations may optionally handle the self-referential scenario, however most current implementations do not do so.

This interface is a member of the Java Collections Framework.

Since:

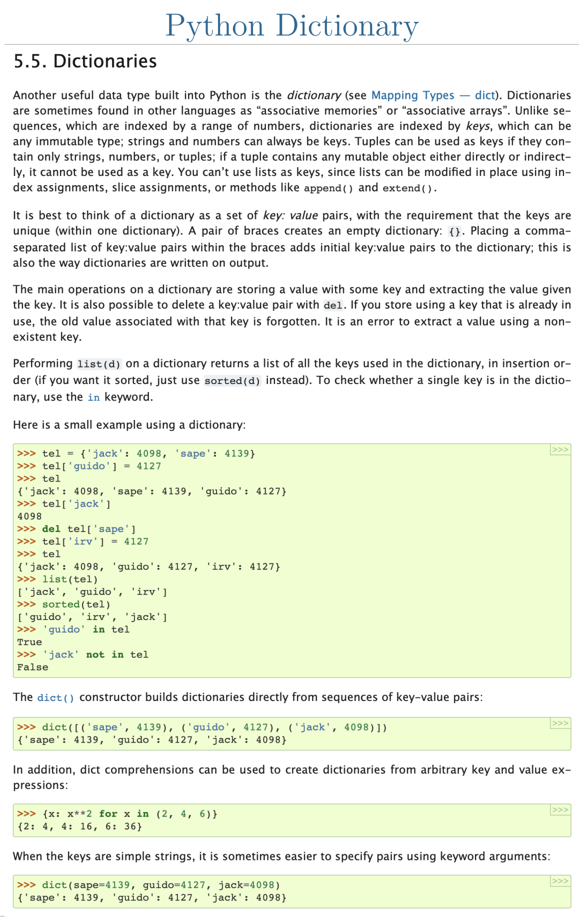
1.2

See Also:

`HashMap`, `TreeMap`, `Hashtable`, `SortedMap`, `Collection`, `Set`

Nested Class Summary

Nested Classes



Python Dictionary

5.5. Dictionaries

Another useful data type built into Python is the *dictionary* (see Mapping Types — dict). Dictionaries are sometimes found in other languages as "associative memories" or "associative arrays". Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. You can't use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like `append()` and `extend()`.

It is best to think of a dictionary as a set of *key: value* pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: `{}`. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with `del`. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

Performing `list(d)` on a dictionary returns a list of all the keys used in the dictionary, in insertion order (if you want it sorted, just use `sorted(d)` instead). To check whether a single key is in the dictionary, use the `in` keyword.

Here is a small example using a dictionary:

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

The `dict()` constructor builds dictionaries directly from sequences of key-value pairs:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

In addition, dict comprehensions can be used to create dictionaries from arbitrary key and value expressions:

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

When the keys are simple strings, it is sometimes easier to specify pairs using keyword arguments:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

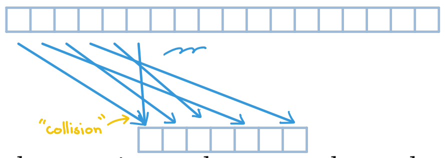
Figure 3.1: The `Map` interface in `Java` (left) and the built-in `Dictionary` data structure in `Python` (right).

size $n+1$ and copying everything over. It's more problematic when the keys are not a neat contiguous sequence from 1 to n. Perhaps the indices arise implicitly in the bit-string representation of some text, in which case these indices will be spread out over a huge range of possible keys. One cannot allocate an array so big. One could alternatively reindex (i.e., rename) the n arbitrary keys into the slots $1, \ldots, n$. This works in *static* situations where the keys are presented at the beginning and never change thereafter. But recall that the primary appeal of dictionaries is their flexibility, and their ability to handle all sorts of different keys, without foresight.

A deterministic way to implement dictionaries is via search trees. If the keys are comparable (such as numbers, or strings in alphabetical order), then search trees can organize the data in sorted order in a tree-like data structure. With a well-designed search tree, searching for a key has roughly the performance of a binary search over a sorted array: $O(\log n)$ time per `get` and `set`. These data structures are often ingenious. Red-black trees use one-bit markers at each node to detect if a subtree has become too "tilted" in one way or another, and rebuilds the tilted portion whenever this occurs. Lazy rebuilding explicitly counts the number of keys in each subtree, and rebuilds an entire subtree when one child subtree becomes much larger than the other (a.k.a. "scapegoat trees" [And89; GR93]). The celebrated *splay tree* data structure by Sleator and Tarjan [ST85b] readjusts itself with every `get` and `set` operation and achieves $O(\log n)$ amortized time (i.e., $O(k \log n)$ time for any sequence of k operations). Another deterministic approach to dictionaries is *tries*, which requires the keys to be (fairly short) bit strings, and uses each successive bit to dictate which direction to go down a binary tree. By compressing long paths in these trees (such as in Patricia tries [Mor68]), these algorithms can be compact and efficient. Now, as clever as these data structures are, they suffer some drawbacks compared to arrays. The $O(\log n)$ query time for search trees is a bit higher then the $O(1)$ time of arrays[2]. They are more complicated to implement, and require a lot of pointer chasing, generating cache misses on the CPU.[3]

We instead consider simpler *randomized* approaches to the dictionary problem; namely, *hash tables*. Hash tables combine the dynamic flexibility of search trees with the raw efficiency of arrays. The only drawback is that the performance guarantees are randomized, which requires a little more sophistication in the analysis. But most people consider the net tradeoff to easily be worth it.

Hash tables are generally based on the following framework. Suppose that there are n keys $k_1, \ldots, k_n$ from the set of integers $[U] = \{1, \ldots, U\}$, where U is typically incredibly large. One allocates an array $A[1..m]$ of size m (typically $m = O(n)$), and randomly constructs a *hash function* $h : [U] \to [m]$.



Ideally, each key-value pair (k_i, v_i) is stored in the slot $A[h(k_i)]$. The remaining question is what to do when keys *collide*, i.e., when $h(k') = h(k'')$ for two distinct keys k' and k''. There are various ways, to account for collisions, sometimes simple and sometimes clever, such as the following.

1. Make ℓ so large that even a single collision is unlikely. Exercise C.6 studies how large m needs to be (relative to n) for this to occur.

2. For each slot $j \in [\ell]$ in the hash table, build a linked list of all keys that hash to

[2]Sometimes, log factors matter.
[3]This last point can be helped to some extent by *cache-oblivious* versions.

slot j. We study this first in Section 3.2.

3. For each slot $j \in [\ell]$ in the hash table, build a second hash table (this time following strategy 1) for all keys that hash to slot j. This is the topic of exercise C.63.

4. Suppose we want to insert a key k. Make *two* hash keys, $h_1(k)$ and $h_2(k)$, and hope that one of these two hash keys is open. More radically, if $h_1(k)$ and $h_2(k)$ are occupied by other keys, see if it is possible to move one of these other keys to its own extra hash key, possibly bumping more keys recursively. This wild approach is called *cuckoo hashing*.

5. Suppose we want to insert a key k and $A[h(k)]$ is occupied. Start scanning the array $A[h(k)+1], A[h(k)+2], \ldots$ until we find the first empty slot, and put k there instead. This approach is called *linear probing*, and will be the topic of the second half of our discussion.

These hash tables have the appeal of potentially running in *constant time*, like an array. Given a key, the hash code $h(k)$ gives a direct index into an array. If the key is there, then we are done. While there may be collisions, we can see in each of the strategies above that $A[h(k)]$ still gets us very "close" to the final location of k. Maybe we have to traverse a short list, hash into a secondary hash table, or continue to scan A until we find our key. For each of these algorithms, some probabilistic analysis is required to understand how much time the "collision-handling" stage will take.

One final remark about the size of hash tables: above, we acted as if we knew *a priori* the number of keys that will be put in the table, and used this to choose the size of the array A. Sometimes, that is the case, but oftentimes it is not, and again the point of dictionary data structures is to not have to plan for these things ahead of time. The easy way to handle an unknown number of keys is by the *doubling trick*. We start with 0 keys and a modestly sized array A; say, of size 64. Whenever the number of keys approaches a constant fraction of the capacity (say, 16), we double the size of the array (to 128). This means we allocate a new array A' with double the capacity, scan the previous array A, and rehash each of the items into A'. A simple amortized analysis shows that the extra effort spent rebuilding is neglible. We note that there are some *distributed* computational settings where one wants to maintain a *distributed dictionary*, and where simply rehashing items becomes expensive and impractical. We refer the reader to a technique called *consistent hashing* that addresses this challenge [KLL+97]. Distributed dictionaries are particularly useful for caching on the web.

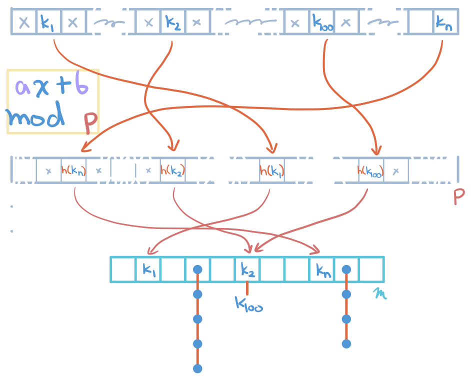


Figure 3.2: Hash tables with universal hashing and chaining.

3.2 Hash tables with chaining

We first consider hash tables that use linked lists to handle collisions. These are maybe the easiest to analyze, and also are most similar in spirit to the `count-min` data structure from Chapter 2.

We recall the basic framework. We have n distinct keys $k_1, \ldots, k_n$, from a universe of integers $\{1, \ldots, U\}$. We allocate an array $A[1..m]$ of size m. (Eventually we will set $m = O(n)$, but for the moment we leave it as a variable to explore the tradeoffs between smaller and larger m.)

We randomly construct a hash function $h : [U] \to [m]$. Here we analyze the setting where h is a universal hash function, but later we will also explore stronger notions of independence. Exercise C.1 explores the setting where h is an ideal hash function.

We hash the n keys into A. At each slot $A[i]$, we build a linked list over all the keys k_j such that $h(k_j) = i$. To find a key k, we go to the linked list stored at $A[h(k)]$, and scan the linked list looking for key k. A high level diagram of the scheme is given in Fig. 3.2. Clearly, the running time of each `get` and `set` will be proportional to the length of the list at the hashed array index. Thus most of our analysis will focus on the lengths of these lists.

We first recall the definition of a *universal hash function.*

Definition 3.1. *A randomly constructed function $h : [n] \to [m]$ is* universal *if, for any two indices $i_1 \neq i_2$, we have*

$$\mathbf{P}[h(i_1) = h(i_2)] = \frac{1}{m}.$$

We also remind the reader that a universal hash function can be constructed as a random function of the form $h(x) = (ax + b \mod p) \mod m$, where p is a prime number larger than the maximum possible key.

Now we present the expected running time of a hash table with chaining and universal hash functions, as a function of m and n. We encourage the reader to attempt the proof themselves.

Theorem 3.2. *Consider chaining with n keys, an array $A[1,...,m]$, and a universal hash function $h : [U] \to [m]$. Then each* `get` *and* `set` *takes $O(1 + n/m)$ time in expectation. In particular, for $m = O(n)$, hash tables with chaining takes $O(n)$ total space and $O(1)$ time per operation in expectation.*

Proof. The time to insert a key k is proportional to the number of collisions with k (plus $O(1)$). The expected number of collisions

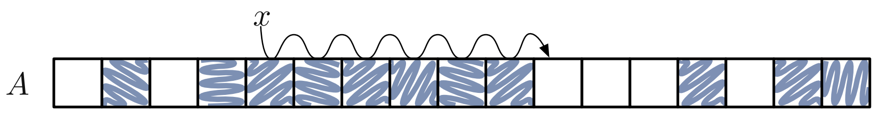
$$\mathbf{E}[|k' : h(k') = h(k)|] \overset{\text{(a)}}{=} \sum_{k' \neq k} \mathbf{P}[h(k') = h(k)] \overset{\text{(b)}}{=} \sum_{k' \neq k} \frac{1}{m} \overset{\text{(c)}}{=} \frac{n-1}{m}$$

Here (a) is by linearity of expectation. (b) is by universality. (c) is because there are $n - 1$ other keys. □

3.3 Linear probing

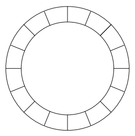
In this section, we explore a different strategy for handling collisions that is arguably more natural: if a key finds its hashed slot already occupied, find the next empty slot in the array and put it there instead.

The hash table, like before, consists of an array $A[1, \ldots, m]$ and a hash function $h : \{1, \ldots, U\} \to \{1, \ldots, m\}$. To insert an item x, we first try to place x at $A[h(x)]$. If $A[h(x)]$ is already occupied, then we instead find the next unoccupied index in the array and place x there instead. (If we reach the end of the array A, then we wrap around to $A[1]$ and continue.)



Since an item x is not necessarily stored at its hashed cell $A[h(x)]$, we carefully use the following terminology. We say that an item *hashes* to a cell $A[i]$ if $h(x) = i$. We say that item x *occupies* a cell $A[i]$ if $A[i] = x$. We stress that an item x hashing into a cell $A[i]$ *does not imply* that x occupies $A[i]$, and that an item x occupying a cell $A[i]$ *does not imply* that x hashes to $A[i]$.

Given two indices $a, b \in [m]$, we define the *interval from a to b*, denoted $[a, b]$, to be the set of indices $\{a, a + 1, \ldots, b \mod m\}$. The "mod m" means that if $b < a$, then we wrap around: $[a, b] = \{a, a + 1, \ldots, m, 1, \ldots, b\}$. One might imagine the array A arranged in a circle rather than a line.



Lemma 3.3. *If an item x occupies cell $\ell \in [m]$, then all of the cells in the interval $[h(x), \ell]$ are occupied.*

Proof. The invariant holds initially with an empty array. We maintain the invariant in the lemma with each insertion, as we insert x in the next unoccupied cell starting from $h(x)$. □

Lemma 3.3 justifies the following lookup procedure. To look up an item x, we first check entry $A[h(x)]$. If item x is not there and the slot is empty, then we conclude the item is not in the array. If the slot $A[h(x)]$ is occupied, but occupied by some item other than x, then we start scanning the array cells to the right of $A[h(x)]$ for either item x or any empty cell. If we find an empty slot before finding x, then by Lemma 3.3, it must be that x is not in the hash table.

To delete an item x, we first find it by the same process as when looking up: starting from $A[h(x)]$, we start scanning the cells until we find x. When we find x at some cell i, we delete x from the cell, but then to restore the invariant in Lemma 3.3, we look for another item to try to fill it. In particular, we start scanning the cells for the first item x_1 with $h(x_1) \leq i$, or else an empty cell. If we find such an item x_1 in a cell i_1, then we put it in the cell i where x was deleted from. We then continue scanning for an item to replace i_1, and so forth.

This hashing scheme is called *linear probing*, and has a special place in the history of computer science. It was analyzed by Donald Knuth in 1962 [Knu63]. Knuth has been called the "father of the analysis of algorithms", and he is credited with formalizing the subject and popularizing O-notation.[4] As Knuth this was the first algorithm he ever formally analyzed[5]. He showed that for ideal hash functions, the expected time of any operation is $O((1/(1-\rho))^2)$ for $\rho = n/m$; in particular, a constant, whenever m exceeds n by a constant factor. This data structure also works very well in practice, even if hash functions in practice are not truly independent. Part of that is owed to the simplicity of the data structure. Scanning an array is extremely fast on hardware, and much faster than chasing pointers along a linked list.

[4]He also invented TeX, solved many problems in compiler design, invented many other important algorithms, wrote *The Art of Computer Programming*, and much more... see for example his wikipedia page.

[5]See for example this interview: `https://www.youtube.com/watch?v=Wp7GAKLSGnI`.

Post-Knuth, there remained a question of how much independence was required to get a constant running time in expectation. We say that a hash function $h : [U] \to [m]$ is *k-wise independent* for $k \in \mathbb{N}$ if for any k distinct keys $x_1, \dots, x_k \in [U]$, and any k values $v_1, \dots, v_k \in [m]$, we have

$$\mathbf{P}[h(x_1) = v_1 \wedge h(x_2) = v_2 \wedge \cdots \wedge h(x_k) = v_k] = \frac{1}{m^k}.$$

That is, the hash values of any fixed set of k (or fewer) keys behaves as if they were produced by an ideal hash function.

(Construct a k-wise independence is more subtle than constructing a universal hash function (like in exercise 2.2), but ultimately can be sampled efficiently in $O(k \operatorname{poly}(\log(n), \log(m)))$ time and represented in $O(k(\log(n) + \log(m)))$ bits. See [Vad12, Chapter 3].)

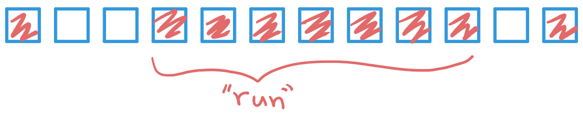
For what values of k does linear probing, with k-wise independent hash function and $m = O(n)$, run in $O(1)$ expected time? Around 1990, Schmidt and Siegel [SS89; SS90] showed that $O(\log n)$-wise independence sufficed[6]. Then, in 2007, Pagh, Pagh, and Ruzic [PPR09] showed that (just!) 5-wise independence sufficed. This was dramatic progress for arguably the oldest problem in algorithm design. Soon after, [PT16] showed that 4-wise independence was *not* enough. So the answer is 5!

Here we give a simplified analysis of the result of [PPR09] based on ideas in [PT16]. We don't put too much emphasis on the constants, preferring to keep the main ideas as clear as possible. Much better constants can be found in [PPR09] and also the reader is encouraged to refine the analysis themselves. Similar proofs of the constant time bound can be found in [Tho15b; Nel16].

3.3.1 Warmup: Ideal hash functions

Consider linear probing with ideal hash functions, assuming $m \geq 2en$.

Suppose we search (or insert or delete) an item x. Each operation on an item x takes time proportional to the number of consecutive occupied cells starting from $A[h(x)]$. To help analyze this length, we introduce the notion of "runs".



A *run* is defined as a maximal interval of occupied slots. Every occupied cell is contained in a unique run. If an item x is in the hash table, then $A[h(x)]$ is occupied,

[6]Alan Siegel taught *me* algorithms.

and x occupies a cell in the run containing $A[h(x)]$. Therefore each operation with an item x takes time bounded by the length of the run containing x. The key question is: what is the expected size of the run at $h(x)$?

Let R represent the (randomized) run containing $h(x)$ and let $|R|$ denote its length. We have

$$\mathbf{E}\begin{bmatrix}\text{running time} \\ \text{(up to constants)}\end{bmatrix} \leq \mathbf{E}[|R|] = \sum_{\ell=1}^{n} \ell\, \mathbf{P}[|R| = \ell]$$

Fix a particular length ℓ and consider the probability that R has length ℓ. There are ℓ possible intervals at $h(x)$, sliding from the interval $[h(x)+1-\ell, h(x)]$ ending at $h(x)$ to the interval $[h(x), h(x)+\ell-1]$ starting from $h(x)$.

Let I be one of these intervals containing $h(x)$. We have $R = I$ only if exactly ℓ of the n elements are hashed into the ℓ slots in I. (Why?) The latter occurs with probability at most

$$\mathbf{P}[R = I] \leq \binom{n}{\ell}\left(\frac{\ell}{m}\right)^{\ell} \overset{\text{(a)}}{\leq} \left(\frac{ne}{m}\right)^{\ell} \leq 1/2^{\ell},$$

where (a) uses the inequality $\binom{n}{\ell} < (ne/\ell)^{\ell}$.

There are ℓ intervals of length ℓ, and each equals R with probability at most $1/2^{\ell}$. By the union bound,

$$\mathbf{P}[|R| = \ell] \leq \frac{\ell}{2^{\ell}}.$$

Altogether, the expected length of R is

$$\mathbf{E}[|R|] = \sum_{\ell} \frac{\ell^2}{2^{\ell}} = O(1),$$

as desired.

3.3.2 Technical preliminaries: 4-wise independence

Our goal now is to establish the same constant time running time for hash functions with limited independence. This analysis will require a particular *concentration inequality* for 4-wise independence hash functions. Here we only state a definition and the lemma statement; a more detailed discussion and proof is given in Section 3.4.

Definition 3.4. *A collection of n variables $X_1, \dots, X_n$ is k-*wise independent *if for any k variables $X_{i_1}, \dots, X_{i_k}$, and values $y_1, y_2, \dots, y_k$, we have*

$$\mathbf{P}[X_{i_1} = y_1, X_{i_2} = y_2, \cdots, X_{i_k} = y_k] = \mathbf{P}[X_{i_1} = y_1]\,\mathbf{P}[X_{i_2} = y_2] \cdots \mathbf{P}[X_{i_k} = y_k].$$

Thus a k-wise independent hash family is one where the hash values are k-wise independent.

In our applications, we will be interested in 5-wise independent hash functions. In the analysis, we will encounter sums of 4-wise independent random variables. The following lemma will be very important.

Lemma 3.5. *Let $X_1, X_2, \dots, X_n \in \{0, 1\}$ be 4-wise independent random variables where $\mathbf{P}[X_i = 1] = p$ for each i. Let $\mu = pn$ be the expected sum and assume $\mu \geq 1$. Then for all $\beta > \mu$,*

$$\mathbf{P}\left[\sum_{i=1}^{n} X_i \geq \mu + \beta\right] \leq \frac{4\mu^2}{\beta^4}.$$

To develop some intuition, let us compare the lemma above to Markov's inequality. Let $X_1, \dots, X_n$ and μ be as in Lemma 3.5. Markov's inequality say that for all $\alpha > 0$,

$$\mathbf{P}[X_1 + \cdots + X_n \geq (1 + \alpha)\mu] \leq \frac{1}{1 + \alpha}. \tag{3.1}$$

Lemma 3.5 says that

$$\mathbf{P}[X_1 + \cdots + X_n \geq (1 + \alpha)\mu] \leq \frac{4}{\alpha^4 \mu^2} \tag{3.2}$$

Compare the RHS of (3.1) with the RHS of (3.2). In (3.2), the upper bound is decreasing in α at a $1/\alpha^4$ rate, compared $1/\alpha$ in (3.1). Moreover, (3.2) is decreasing in the expected value μ at a rate of $1/\mu^2$. That is, the *greater the mean*, the *smaller the probability of deviated from the mean.* This is an example of a *concentration inequality.* We will soon see why this helpful in the analysis of linear probing.

3.3.3 Analysis of linear probing with 5-wise independence.

Theorem 3.6. *Let h be 5-wise independent. For $m \geq 8n$, linear probing takes expected constant time per operation.*

Proof. As discussed in Section 3.3.1, the running time of an operation on an item x is bounded by the length of the run R containing $h(x)$. Let $i = h(x)$, and let R be the run at index i. As before, we have

$$\mathbf{E}\begin{bmatrix}\text{running time}\\ \text{(up to constants)}\end{bmatrix} \leq \mathbf{E}[|R|] = \sum_{\ell=1}^{n} \ell\, \mathbf{P}[|R| = \ell],$$

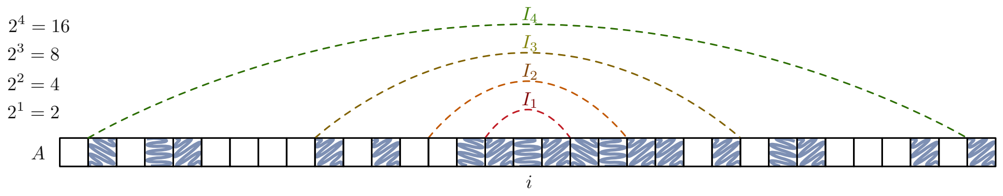
and we want to bound $\mathbf{P}[|R| = \ell]$ for each ℓ. Here we apply a "doubling trick" and group the run lengths by powers of 2:

$$\mathbf{E}\begin{bmatrix}\text{running time}\\ \text{(up to constants)}\end{bmatrix} \leq \sum_{\ell=1}^{n} \ell\, \mathbf{P}[|R| = \ell] \leq \sum_{k=1}^{\lceil \log n \rceil} 2^k\, \mathbf{P}\Big[2^{k-1} < |R| \leq 2^k\Big]. \tag{3.3}$$

We fix $k \in \mathbb{N}$ and analyze the event that $2^{k-1} < |R| \leq 2^k$. Let

$$I_k = [i - (2^k - 1), i + 2^k - 1]$$

be the interval of length $2^{k+1} + 1$ centered at i.



Observe that all the items occupying R were also hashed into R. Moreover, if R has length $|R| < 2^k$ and contains i, then R is contained in I_k. Thus, for each k, we have

$$\mathbf{P}\Big[2^{k-1} < |R| \leq 2^k\Big] \leq \mathbf{P}\begin{bmatrix}\text{at least } 2^{k-1} \text{ other}\\ \text{items hash into } I_k\end{bmatrix}.$$

Let

$$\mu \overset{\text{def}}{=} \mathbf{E}\begin{bmatrix}\#\text{ other items}\\ \text{hashing into } I_k\end{bmatrix}.$$

Since h is 5-wise independent, conditional on $h(x) = i$, the remaining hash values are 4-wise independent. Each lands in I_k with probability $p = |I_k|/m$. We have

$$\mu = \frac{|I_k| n}{m} \overset{\text{(a)}}{\leq} 2^{k-2},$$

where (a) is because $m \geq 8n$. We have

$$\mathbf{P}\left[\begin{pmatrix}\text{\# other items}\\ \text{hashing into } I_k\end{pmatrix} > 2^{k-1}\right] \overset{\text{(b)}}{\leq} \frac{4\mu^2}{\left(2^{k-1}-\mu\right)^4} \leq \frac{4\left(2^{k-2}\right)^2}{\left(2^{k-2}\right)^4} \leq \frac{1}{2^{2k-6}}.$$

Here (b) is by Lemma 3.8. Plugging back into RHS(3.3) above, we have

$$\mathbf{E}\left[\substack{\text{running time}\\ \text{(up to constants)}}\right] \leq \text{RHS(3.3)} \leq \sum_{k=1}^{\lceil \log n\rceil} 2^k \cdot \frac{1}{2^{2k-6}} = 2^6 \sum_{k=1}^{\lceil \log n\rceil} \frac{1}{2^k} \leq 2^6.$$

A constant! □

3.4 4-wise independence

We close the chapter with some probabilistic analysis of k-wise independent random variables. In particular we prove Lemma 3.5, which played a key role in the analysis of linear probing.

3.4.1 Expectations of products of k-wise independent families

Recall the definition of k-wise independent random variables. The following lemma observes that the expected value of a product of (at most) k, k-wise independent random variables is the product of the values.

Lemma 3.7. *Let $X_1, \dots, X_k$ be k-wise independent random variables. Then*

$$\mathbf{E}[X_1 X_2 \cdots X_k] = \mathbf{E}[X_1]\,\mathbf{E}[X_2]\cdots\mathbf{E}[X_k].$$

Before proving Lemma 3.7, let us give a simple example where k-wise independence matters. Let $X_1, \cdots, X_k \in \{0,1\}$ where each X_i denotes the outcome of a fair coin toss - 0 for tails, 1 for heads. Then $X_1 \cdots X_k = 1$ if all of the coin tosses come up heads, and 0 otherwise. Consider the following parallel universes.

1. Suppose each X_i was based on a different, independent coin toss. That is, $X_1, \dots, X_k$ are mutually independent. The probability that k independent coin tosses all comes up heads is $1/2^k$, so $\mathbf{E}[X_1 \cdots X_k] = 1/2^k$.

2. Suppose each X_i was based on the *same* coin toss. That is, $X_1 = \cdots = X_k$; they are certainly *not* k-wise independent. Then the probability that all $X_1, \dots, X_k = 1$ is the probability of a single coin coming up heads, 1/2, and so $\mathbf{E}[X_1 \cdots X_k] = 1/2$.

Here there is an exponential gap between independent and non-independent coin tosses.

Proof of Lemma 3.7. We have

$$\begin{aligned}
&\mathbf{E}[X_1 X_2 \cdots X_k] \\
&\overset{\text{(a)}}{=} \sum_{y_1, y_2, \ldots, y_k} y_1 y_2 \cdots y_k \, \mathbf{P}[X_1 = y_1,\, X_2 = y_2,\, \ldots, X_k = y_k] \\
&\overset{\text{(b)}}{=} \sum_{y_1, y_2, \ldots, y_k} y_1 y_2 \cdots y_k \, \mathbf{P}[X_1 = y_1] \, \mathbf{P}[X_2 = y_2] \cdots \mathbf{P}[X_k = y_k] \\
&= \left(\sum_{y_1} y_1 \, \mathbf{P}[X_1 = y_1] \right) \left(\sum_{y_2} y_2 \, \mathbf{P}[X_2 = y_2] \right) \cdots \left(\sum_{y_k} y_k \, \mathbf{P}[X_k = y_k] \right) \\
&\overset{\text{(c)}}{=} \mathbf{E}[X_1] \, \mathbf{E}[X_2] \cdots \mathbf{E}[X_k].
\end{aligned}$$

Here (a) is by definition of expectation[7]. (b) is by k-wise independence. (c) is by definition of expectation, for each X_i. □

3.4.2 A concentration inequality for 4-wise independent sums.

Now we prove Lemma 3.5. Below the claim is stated slightly more generally than in Lemma 3.5.

Lemma 3.8. *Let $X_1, X_2, \ldots, X_n \in \{0, 1\}$ be 4-wise independent variables where for each i, $\mathbf{E}[X_i] = p$. Let $\mu = pn = \mathbf{E}[\sum_{i=1}^n X_i]$. Then for any $\beta > 0$,*

$$\mathbf{P}\left[\sum_{i=1}^n X_i \geq \mu + \beta \right] \leq \frac{\mu + 3\mu^2}{\beta^4}.$$

Proof. We have

$$\mathbf{P}\left[\sum_{i=1}^n X_i \geq \mu + \beta \right] = \mathbf{P}\left[\sum_{i=1}^n X_i - \mu \geq \beta \right] \quad \overset{\text{(a)}}{\leq} \mathbf{P}\left[\left(\sum_{i=1}^n X_i - \mu \right)^4 \geq \beta^4 \right] \overset{\text{(b)}}{\leq} \frac{\mathbf{E}\left[(\sum_{i=1}^n X_i - \mu)^4 \right]}{\beta^4}.$$

The key step is (a), where we raise both sides to the fourth power. (b) is by Markov's inequality. *We claim that*

$$\mathbf{E}\left[\left(\sum_{i=1}^n X_i - \mu \right)^4 \right] \leq \mu + 3\mu^2,$$

[7]We are summing over all possible outcomes $(y_1, \ldots, y_k)$ of $(X_1, \ldots, X_k)$, multiplying the value, $y_1 \cdots y_k$, with the probability of the outcome, $\mathbf{P}[X_1 = y_1, \ldots, X_k = y_k]$.

which would complete the proof. We first have

$$\mathbf{E}\left[\left(\sum_{i=1}^{n} X_i - \mu\right)^4\right] = \mathbf{E}\left[\left(\sum_{i=1}^{n}(X_i - p)\right)^4\right]$$

because $\mu = pn$. Now, $(\sum_{i=1}^{n}(X_i - p))^4$ expands out to the sum

$$\sum_{i=1}^{n}(X_i - p)^4 + \binom{4}{2}\sum_{i<j}(X_i - p)^2(X_j - p)^2 + \binom{\text{monomials w/ some}}{(X_i - p) \text{ w/ degree } 1}. \tag{3.4}$$

Some examples of the third category would be $(X_1 - p)^3(X_2 - p)$, $(X_1 - p)^2(X_2 - p)(X_3 - p)$, and $(X_1 - p)(X_2 - p)(X_3 - p)(X_4 - p)$. Consider the expected value of each of these categories of monomials.

1. For each i, we have

$$\mathbf{E}\Big[(X_i - p)^4\Big] = p(1-p)^4 + (1-p)p^4 p(1-p).$$

2. For each $i \neq j$, we have

$$\mathbf{E}\Big[(X_i - p)^2(X_j - p)^2\Big] \overset{(c)}{=} \mathbf{E}\Big[(X_i - p)^2\Big]\,\mathbf{E}\Big[(X_j - p)^2\Big] \overset{(d)}{\leq} p^2(1-p)^2.$$

   Here (c) is because of pairwise independence. (d) is because

$$\mathbf{E}\Big[(X_i - p)^2\Big] = p(1-p)^2 + (1-p)p^2 \leq p(1-p).$$

3. Each monomial in the third category has expected value 0. This is because we can pull out the degree 1 term by independence, which has expected value 0. For example,

$$\mathbf{E}\Big[(X_1 - p_1)^3(X_2 - p_2)\Big] \overset{(e)}{=} \mathbf{E}\Big[(X_1 - p_1)^3\Big]\,\mathbf{E}[X_2 - p_2] \overset{(f)}{=} 0,$$

   where (e) is by pairwise independence, and (f) is because $\mathbf{E}[X_2 - p_2] = 0$.

Plugging back in above, we have

$$\mathbf{E}\left[\left(\sum_{i=1}^{n} X_i - \mu\right)^4\right] = np(1-p) + \binom{n}{2}\binom{4}{2}\Big(p^2(1-p)^2\Big) \leq np + 3(np)^2,$$

as desired. This completes the proof. □

Remark 3.9. The claim would hold even for X_i not identically distributed (as long as they are 4-wise independent and are each in $[0, 1]$). The restrictive assumptions here simplify the exposition and suffice for our applications.

3.5 Takeaways

- Dictionary data structures provide everyday motivation for studying randomization, where hash tables offer simpler and better performance (in expectation) than search trees.
- There are different ways to implement hash tables and they mostly differ in how they handle collisions.
- *Chaining* uses linked lists to handle collisions. It has reasonable performance *in expectation* for universal hash functions, and stronger guarantees when the hash function is more independent.
- *Linear probing* is perhaps the easiest hash table to implement, and scanning an array is hardware-friendly. It had been observed to perform well in practice long before it had been properly analyzed.
- The analysis of linear probing cleverly uses *canonical intervals* (doubling in size) to limit the number of "bad events" we have to avoid, to roughly $\log n$ (per key).
- It turns out that *5-wise independence* is sufficient for linear probing to have $O(1)$ running time in expectation. Interestingly, 4-wise independence is not enough.

3.6 Additional notes and materials

Thorup [Tho15a] describes several families of hash functions with both theoretical and practical considerations. See [Eri17] for additional notes on hashing.

Lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

3.7 Exercises

Exercise 3.1. Let $h : [n] \to [\ell]$ be an ideal hash function, with $\ell \geq n$. What is the *exact probability* that h has no collisions (i.e., h is injective)?

Exercise 3.2. Consider the particular case of hash tables with chaining with $k = n$ and an *ideal* hash function $h : [n] \to [n]$. Let $A[1..n]$ be the cells of the hash table.

1. Consider a particular array slot $A[i]$. Show that for $\ell \in \mathbb{N}$, the probability that $A[i]$ has $\geq \ell$ items hashed to it is
$$\mathbf{P}[\text{at least } \ell \text{ items being hashed to } A[i]] \leq \frac{1}{\ell!}.$$
2. Show that, with probability of error $\leq 1/n^2$, the maximum length is at most $O(\log(n)/\log\log n)$.[8]

Exercise 3.3. The goal of this exercise is to show how to get constant time access for n keys with $O(n)$ space, using only universal hash functions. We will require the following fact that we ask you to prove.

1. Let $h : [n] \to [k]$ be a *universal* hash function. Show that for $k \geq n^2$, h has no collisions with probability $\geq 1/2$.

Now we describe the data structure. We first allocate an array $A[1..n]$ of size n. We have one universal hash function h_0 into $[n]$. If we have a set of (say) k collisions at an array cell $A[i]$, rather than making a linked list of length k, and we build another hash table, with a new universal hash function h_i, of size k^2, with no collisions (per part 1). (We may have to retry if there is a collision.) If the total size (summing the lengths of the first arrray and each of the second arrays) comes out to bigger than (say) $5n$, we try again.

[8]The simple lower bound of $\ell! \geq (\ell/2)^{\ell/2}$ may be helpful. It is implicit in the $O(\cdots)$ notation that your bound need only hold for n sufficiently large.

2. For each $i = 1, \ldots, n$, let k_i be the number of keys that hash to the ith cell. We have

$$\text{(sum of array sizes of our data structure)} \leq n + \sum_{i=1}^{n} k_i^2.$$

Show that[9]

$$\sum_{i=1}^{n} k_i^2 \leq n + O(\text{total \# of collisions (w/r/t } h_0)).$$

3. Show that

$$\mathbf{E}[\text{total \# of collisions (w/r/t } h_0)] \leq n/2.$$

4. Show that

$$\mathbf{P}[\text{(sum of all array sizes)} > Cn] \leq 1/2$$

for some constant $C > 0$. ($C = 5$ is possible.)

Taken together, steps 1 to 3 above show that this approach will build a "perfect" hash table over the n keys in $O(n)$ space with probability of success at least $1/2$, using only universal hash functions. Even if it fails to work, we can then keep repeating the construction until it succeeds. This approach works better in *static settings*, when the set of keys is fixed.

[9]Here a "collision" is an unordered pair of keys with the same hash. The $O(\cdots)$ means you can choose whatever constant you find convenient; 2 is possible.

Chapter 4

Sampling edges

4.1 Minimum cut

Recall the *minimum cut* problem in undirected graphs. The input consists of a connected, undirected graph $G = (V, E)$ with positive edge capacities $c : E \to \mathbb{R}_{>0}$. A *cut* is a set of edges $C \subseteq E$ whose removal disconnects the graph. The goal is to

$$\text{minimize } \sum_{e \in C} c(e) \text{ over all cuts } C \subset E. \tag{4.1}$$

This problem is polynomial time solvable. Whatever the optimum cut is, it must be a minimum (s, t)-cut for some pair of vertices s and t. Thus to find the global minimum, one can guess s and t by looping over V, and compute the minimum $\{s, t\}$-cut for each choice of s and t. (Better yet: fix s, and loop over all t.)

For a set of vertices S, let $\delta(S)$ denote the set of edges with exactly one endpoint in S. $\delta(S)$ is called the cut *induced by* S. The induced cuts are also the inclusionwise minimal cuts, and it suffices to consider only the induced cuts when solving (Min-Cut).

We will study a subtle algorithm discovered by Karger [Kar93], that has been influential beyond the minimum cut problem. Consider the following description of Karger's algorithm.

> *Repeatedly sample edges in proportion to their capacities until there is only one cut from which we have not yet sampled any edges. Return this cut.*

This algorithm is clearly ridiculous. For the unweighted setting, the above algorithm is equivalent to the following, equally absurd approach (see exercise C.52).

> *Independently assign every edge $e \in E$ a weight $w_e \in [0, 1]$ uniformly at random. Build the minimum weight spanning tree T w/r/t w. Let e be the heaviest edge in T. Return the cut induced by the two components of $T - e$.*

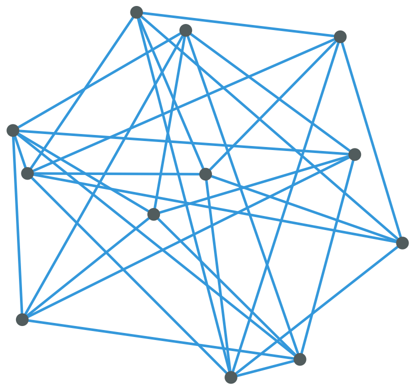
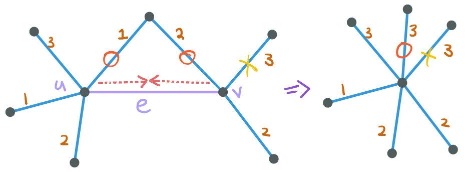


Figure 4.1: What is the minimum cut in this graph?

Compare the two approaches above. Of course we know how to compute the minimum spanning tree; among other approaches, we can repeatedly add the smallest weight edge to T that does not create a cycle. On the other hand, in the first approach, it might appear difficult to keep track of which cuts we have and have not sampled from, being that there are so many cuts. This can be addressed by *contracting the graph*.



Suppose we sample an edge $e = \{s, t\}$. Then we know that any cut $\delta(S)$, where $s \in S$ and $t \notin S$, has now been sampled from. Thus we can safely *contract* e; replacing s and t with a single vertex u that has the sum[1] of edges incident to s and t. Note that contracting e will only effect cuts that contain e.

Now imagine we contract edges as we sample them. Eventually there are only two vertices left in the contracted graph, which represent two connected components in the input graph. These components induce the only cut we have not yet sampled from, and this is the cut that we return.

Pseudocode for the contraction algorithm is given in Fig. 4.2. Here, for an edge $e \in G$, we let $G/e = (V/e, E/e)$ denote the graph obtained by contracting e, and we let $c/e : E/e \to \mathbb{R}_{>0}$ denote the corresponding capacities. Figure 4.3 sketches a few iterations of the algorithm applied to a barbell graph.

[1]More precisely, for every edge f of the form $\{s, z\}$ or $\{t, z\}$, we create a new edge $\{u, z\}$ with the same capacity. If s and t both have edges to the same vertex z, we can either create two edges from u to z with the appropriate capacities, or make a single edge from u to v with the same capacity. We remove s and t and its incident edges from the graph, replacing them with u and the newly created edges incident to u.

```
random-contractions($G = (V, E)$, $c$)
1. While $|E| > 1$
    A. Sample $e \sim c$
    B. $G \leftarrow G/e$, $c \leftarrow c/e$
2. Let $E = \{e\}$
3. Return the edges in the original graph that contracted to $e$
```

Figure 4.2: A randomized minimum cut algorithm due to Karger [Kar93].

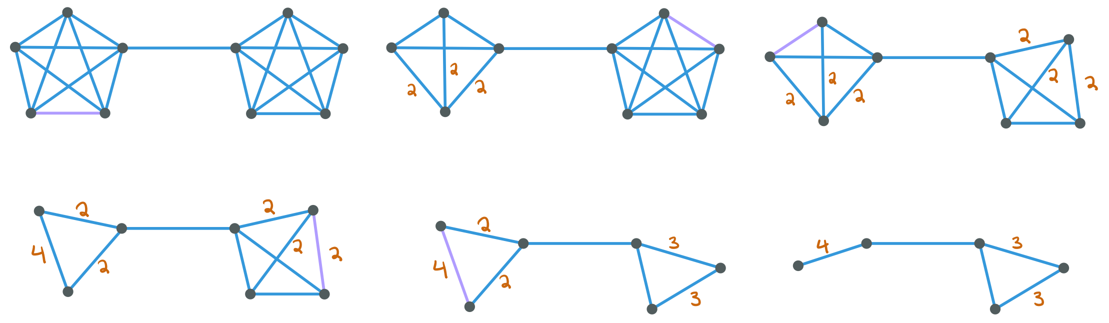


Figure 4.3: A few iterations (from left to right, top to bottom) of the `random-contractions` algorithm applied to a barbell graph.

The intuition behind `random-contractions` is as follows. Here we consider unweighted graphs for simplicity. (The intuition is the same for weighted graphs, except replacing "many edges" with "large capacity", etc.) Suppose we have an unweighted graph $G = (V, E)$, and let $C \subset E$ be the minimum cut. Since C is the minimum cut — keyword minimum — there are presumably very few edges in C. If we randomly sample an edge $e \in E$, then hopefully $e \notin C$. If C "survives" this round, then we have all made some progress because there is one less vertex in the graph after contracting e. In the next round, C is still the minimum cut, so the high-level logic from the first round still holds. Thus we can repeatedly sample edges and preserve the hope that we avoid C.

The above argument hinges on *how much smaller* C is than all of E. If we can argue that C is always a small fraction of E, then that gives hope that C survives to the end. On the other hand, if C is even a small constant fraction of G, we will probably sample from C after a constant number of rounds. Observe also that over time, C becomes a larger and larger fraction of E, as we contract and remove edges outside of C.

The key observation is that *every vertex v induces a cut $\delta(v)$, which must have at least as many edges as C. Thus the minimum cut is at most the minimum degree in the graph.* In turn, since the number of edges in E is the sum of degrees (divided by 2), *the minimum cut C is at most a $2/n$ fraction of the total number of edges*! This observation holds initially in the input graph and thereafter in the contracted graphs, although n decreases by 1 in each iteration.

On the first iteration, C has at most a $2/n$ chance of being hit. On the second interation, assuming C survived the first iteration, C has (at most) a $2/(n-1)$ chance of being hit. Continuing in this fashion, assuming C surivived the first $i-1$ iterations, C has a $2/(n-i+1)$ change of being hit in the ith iteration. If one combines these problems, one discovers that C has a $\geq 1/\binom{n}{2}$ chance of surviving all $n-1$ rounds We can repeat the experiment $\binom{n}{2} = O(n^2)$ (a polynomial!) number of times to find the minimum cut with constant probability, and $O(n^2 \log n)$ times to find the minimum cut with high probability.

In the sequel, we formalize the the above argument, as well as extend it to positive capacities. For ease of notation, for a set of edges $C \subset E$, we denote the sum of capacities over C by

$$\sum_{e \in C} c(e) \overset{\text{def}}{=} \sum_{e \in C} c(e).$$

Lemma 4.1. *Let $C^\star$ be the minimum cut in (G, c), and suppose $e \notin C$. Then $C^\star$ is (or maps to) to the minimum cut in the contracted graph $(G/e, c/e)$.*

Proof sketch. Direct inspection. □

Lemma 4.2. $\sum_{e \in E} c(e) \geq \frac{\lambda n}{2}$.

Proof. Every vertex v has weighted degree $\sum_{e \in \delta(v)} c(e) \geq \lambda$ since $\delta(v)$ is a cut. Thus

$$\sum_{e \in E} c(e) = \frac{\sum_v \sum_{e \in \delta(v)} c(e)}{2} \geq \frac{\lambda n}{2}.$$

□

Lemma 4.3. *Let $e \sim c$. Then* $\mathbf{P}[e \in C^\star] \leq \frac{2}{n}$.

Proof. We have $\mathbf{P}[e \in C^\star] = \frac{\sum_{e \in C^\star} c(e)}{\sum_{e \in E} c(e)} \overset{\text{(a)}}{\leq} \frac{2}{n}$ by (a) Lemma 4.2. □

Lemma 4.4. *Let $C^\star$ be a minimum cut. With probability $\geq 1/\binom{n}{2}$,* `random-contractions` *returns $C^\star$.*

Proof. For $k \in \mathbb{Z}_{\geq 0}$, let E_k be the event that we have not sampled $C^\star$ after k iterations. Initially, $\mathbf{P}[E_0] = 1$, and we want to show that $\mathbf{P}[E_{n-2}] \geq 1/\binom{n}{2}$. By Lemma 4.3, we have

$$\mathbf{P}[E_k \mid E_{k-1}] \geq 1 - \tfrac{2}{n-(k-1)} \text{ for each } k \in [n].$$

The probability of succeeding (event E_{n-2}) is at least

$$\begin{aligned}\mathbf{P}[E_{n-2}] &= \prod_{k=1}^{n-2} \mathbf{P}[E_k \mid E_{k-1}] \geq \prod_{i=3}^{n} \left(1 - \frac{2}{i}\right) \\ &= \prod_{i=3}^{n} \frac{i-2}{i} = \frac{(n-2)!2}{n!} = \frac{1}{\binom{n}{2}}.\end{aligned}$$

□

Thus with probability about $1/n^2$, the random contraction algorithm returns the minimum cut. To find the minimum cut with constant probability, we rerun the algorithm $O(n^2)$ time and return the best cut. To find the minimum cut with *high probability*, we rerun the algorithm $O(n^2 \log n)$ times.

The nice thing about repetition is that we can run the randomized trials *in parallel*. Moreover, a single instance of the contraction algorithm (via its connection to minimum spanning trees) can be made to run in polylog(n) time with polynomially many processors. Thus one obtains a randomized parallel algorithm for minimum cut.

```
branching-contractions($G = (V, E)$, $c$)
1. Let $n = |V|$. If $n \leq 3$ then compute the min-cut by brute force.
2. Until $|V| = \left\lceil \frac{n}{\sqrt{2}} \right\rceil + 1$:
   A. Sample $e \sim c$ and contract $e$ in $G$.
3. $C_1 \leftarrow$ branching-contractions($G$,$c$).
4. $C_2 \leftarrow$ branching-contractions($G$,$c$).
5. Uncontract and return the minimum of $C_1$ and $C_2$.
```

Figure 4.4: A randomized minimum cut algorithm that amplifies the `random-contractions` algorithm by branching, due to Karger and Stein [KS96].

Corollary 4.5. *A randomized minimum cut can be computed in parallel in polylogarithmic time with a polynomial number of processors.*

That said, `random-contractions` is not just an algorithm. It is also a surprising structural observation about the number of minimum cuts in an undirected graph. In the above algorithm, any fixed minimum cut is returned with probability $1/\binom{n}{2}$. This implies that there are at most $\binom{n}{2}$ minimum cuts in the graph!

Corollary 4.6. *There are at most $\binom{n}{2}$ minimum cuts in a graph.*

4.2 Amplification by branching

The `randomized-contraction` algorithm preserves a fixed minimum cut with probability at least $1/\binom{n}{2}$. One can amplify this algorithm directly by running it independently $O(n^2 \log n)$ and outputting the minimum over all the trials. With high probability, the minimum cut lies in one of these $O(n^2 \log n)$ cuts.

Karger and Stein [KS96] reduced the probability of failure more efficiently by an amplification process called *branching*. To motivate the branching technique, recall from Section 4.1 that the probability of failure increases as the number of remaining vertices decreases. Instead of restarting the entire algorithm from the beginning again and again, one might restart the algorithm from some relatively confident point partway through. One might further apply this strategy recursively.

Karger and Stein [KS96] proposed randomly contracting edges as long as the probability of avoiding the min-cut is still at least 1/2, and then "branching"; i.e., running two independent processes that continue from this point. The branching is recursive: each of the two independent trials also continue for a relatively safe number of iterations before branching again. This refined amplification process, it is shown below, is much more efficient than repeated independent trials of `random-contractions`. The amplification technique is interesting in its own right and extends past this particular problem.

A sketch of the algorithm, which we call `branching-contractions`, is given in Fig. 4.4.

Lemma 4.7. *Let $C^\star$, and suppose we contract $n-k$ edges sequentially at random. The probability that none of theses edges samples the minimum cut is at least $\frac{k(k-1)}{n(n-1)}$.*

Proof. For $i \in \mathbb{N}$, let E_i be the probability that we have not sampled $C^\star$ after i iterations. We ware interested in $\mathbf{P}[E_{n-k}]$. We have

$$\begin{aligned}\mathbf{P}[E_{n-k}] &= \prod_{i=1}^{n-k} \mathbf{P}[E_i \mid E_{i-1}] \overset{\text{(a)}}{\geq} \prod_{i=k+1}^{n} \left(1 - \frac{2}{i}\right) \\ &= \frac{\prod_{i=3}^{n} \frac{i-2}{i}}{\prod_{i=3}^{k} \frac{i-2}{i}} = \frac{\binom{k}{2}}{\binom{n}{2}} = \frac{k(k-1)}{n(n-1)}.\end{aligned}$$

by (a) Lemma 4.3. □

Theorem 4.8. `branching-contractions` *runs in $O(n^2 \log n)$, and returns a minimum cut with probability $\Omega(1/\log n)$.*

Proof. We first prove the running time, and then discuss the correctness. The running time is dominated by the recurrence

$$f(n) \leq 2f(\varphi(n)) + n^2 \text{ for } \varphi(n) = n/\sqrt{2} + c$$

for some constant $c > 0$. The recursion tree is drawn in Fig. 4.5. For $i \in \mathbb{N}$, let $\varphi^i = \varphi \circ \varphi^{i-1}$ recursively apply φ i times (e.g., $\varphi^1 = \varphi$). Each problem at depth i has size at most

$$\varphi^i(n) = \left(\frac{1}{\sqrt{2}}\right)^i n + \sum_{j=0}^{i} \left(\frac{1}{\sqrt{2}}\right)^i c \leq \left(\frac{1}{\sqrt{2}}\right)^i n + \frac{c}{\sqrt{2}-1}.$$

Since there are 2^i problems at depth i, the total amount of work at level i is

$$2^i \left(\varphi^i(n)\right)^2 = 2^i \left(\left(\frac{1}{\sqrt{2}}\right)^i n + O(1)\right)^2 = O\left(n^2\right).$$

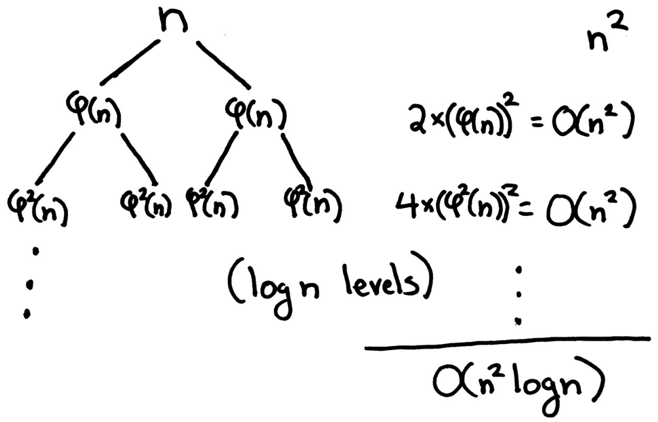


Figure 4.5: A recursion tree analysis for `branching-contractions`.

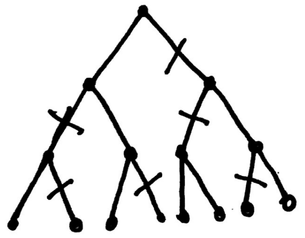
Over $O(\log n)$ levels, then, the total work is $O(n^2 \log n)$.

The proof of correctness is based on a more general phenomena, called the *Galton-Watson process*. We study the Galton-Watson process in greater generality in the following section, and here we only give the reduction from `branching-contractions` to the Galton-Watson process.

We can arrange the recursive calls in a binary tree. Each node consists of a subproblem, with two children consisting of the two subproblems. The leaves are the constant-size subgraphs that can be computed by brute force. The height is $O(\log n)$ because every level decreases n by a constant factor.

The process at a subtree succeeds iff the node succeeds and one of the two subtrees succeeds, and each node succeeds with probability 1/2. The overall algorithm succeeds iff there is a root to leaf path where every node succeeds. This is the Galton-Watson process over $O(\log n)$ generations, which (by Theorem 4.9 below), has a $\Omega(1/\log n)$ probability of success. □

4.3 Randomized branching



Theorem 4.9. *Let T be a complete binary tree of height $k \geq 2$, and suppose every edge is deleted independently with probability $1/2$. The probability that there is a leaf connected to the root is $\geq 1/k$.*

Proof. For $i \in \mathbb{N}$. Let p_i be the probability that a particular node at height i is connected to a subleaf. We have $p_0 = 1$. For a node at height $i+1$, the probability that there is no path to a leaf via a particular child is

$$\frac{1}{2} + \frac{1}{2}(1 - p_i) = 1 - \frac{p_i}{2}.$$

p_{i+1} is 1 one minus the probabilities there is no path to a leaf via either child, which by independence to the two subtrees is

$$p_{i+1} = 1 - \left(1 - \frac{p_i}{2}\right)^2 = p_i\left(1 - \frac{p_i}{4}\right). \tag{4.2}$$

The first three values are

$$p_0 = 1,\ p_1 = 3/4, \text{ and } p_2 = \tfrac{39}{64} \geq 1/2.$$

We claim by induction on k that $p_k \geq \frac{1}{k}$ for all $k \geq 2$. Looking at the RHS of (2), we first observe that function

$$f(x) = x\left(1 - \tfrac{x}{4}\right) \text{ is increasing for } x \leq 2,$$

which can be seen from its derivative $f'(x) = 1 - \frac{x}{2} \geq 0$. In particular, to lower bound p_{k+1} via (2), we can replace p_k by any lower bound for p_k. By induction, $p_k \geq 1/k$, hence

$$p_{k+1} \geq \frac{1}{k}\left(1 - \frac{1}{4k}\right) = \frac{1}{k} - \frac{1}{4k^2} \overset{\text{(a)}}{\geq} \frac{1}{k+1}.$$

The last inequality (a) is obtained by

$$\frac{1}{k} - \frac{1}{k+1} = \frac{1}{k^2 + k} \geq \frac{1}{4k^2} \text{ for } k \geq 1. \tag{4.3}$$

□

4.4 Additional notes and materials

Lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

4.5 Exercises

Exercise 4.1. Consider the randomized algorithm for minimum cut based on building the minimum spanning tree w/r/t randomized weights, described in Section 4.1.

1. Prove that this algorithm is equivalent to the random contractions algorithm for unweighted graphs.

2. Adjust the randomized spanning tree algorithm to account for weights, and prove its correctness.

Exercise 4.2. Let $G = (V, E)$ be an undirected graph. For $k \in \mathbb{N}$ a *k-cut* is a set of edges whose removal disconnects the graph into at least k connected components. Note that for $k \geq 3$, the minimum k-cut problem cannot easily be reduced to (s,t)-flow. In fact, the problem is NP-Hard when k is part of the input.[2]

1. Briefly describe how to modify the `random-contractions` to return a k-cut.[3]

2. Analyze the probability that your modified algorithm returns a minimum k-cut.[4]

[2]You might find it helpful to focus on the case $k = 3$ and then generalize afterwards. Although we ask you to work out the dependency on k, conceptually, it might help to think of k as being relatively small compared to n.

[3]Your algorithm design may be informed by your calculations in part 2.

[4]You may want to pattern your analysis after the one for minimum (2-)cut; in particular, you may want to develop analogs for Lemmas 4.2 and 4.3.

3. Describe and analyze an algorithm, using your modified `random-contractions` as a subroutine, that computes a minimum k-cut with high probability in $O\left(n^{c_1 k} \log^{c_2} n\right)$ time for constants c_1 and c_2. (We leave it to you to identify these constants; as usual, the faster the running time, the better.)

4. How does your algorithm relate to the preceding statement that k-cut is NP-Hard when k is part of the input?

Exercise 4.3. Consider the minimum cut problem in undirected graphs. We say that a cut $C = \delta(S)$ is a 2-*approximate minimum cut* it its capacity is at most twice the capacity of the minimum cut.

1. Let C be an 2-approximate minimum cut. Suppose we run the `random-contractions` algorithm run until there are 5 vertices. Show that C is preserved by the algorithm with probability $\geq 1/\binom{n}{4}$.

2. Show the number of 2-approximate minimum cuts is at most

$$O\left(n^4\right).$$

Exercise 4.4. Consider the minimum cut problem in undirected graphs. For $\alpha \geq 1$, we say that a cut $C = \delta(S)$ is an α-*approximate minimum cut* it its capacity is at most α times the capacity of the minimum cut.

1. Let C be an α-approximate minimum cut. Suppose we run the `random-contractions` algorithm run until there are $O(\alpha)$ vertices remaining. Show that C is preserved by the algorithm with probability $\geq 1/\binom{n}{O(\alpha)}$.

2. Show the number of α-approximate minimum cuts is at most

$$n^{O(\alpha)}.$$

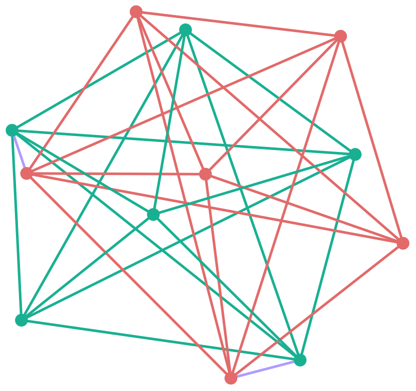


Figure 4.6: The minimum cut from Fig. 4.1

Chapter 5

Random Sums and Graphs

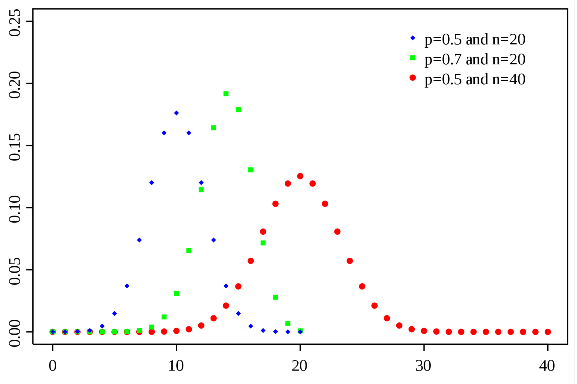
5.1 Random sums

If, out of 100 coin tosses, you were told that 50 of them were heads, would you be surprised? Actually, you should be a little surprised. The odds of getting exactly 50 heads is about 8%. But if you were told that the number was in the range 45 to 55, you probably wouldn't think much of it.

If you were told that all 100 coin tosses came up heads, you wouldn't believe it. The odds of that, we know, is $1/2^{100}$. If you bet money and lost on this event, you would be outraged (and, at even odds, certainly broke for the rest of eternity).

Suppose you were told that at most 25 coin tosses came up heads. Should you be surprised? On one hand, 25 is half of the expected amount. On the other hand, the claim is not that there was exactly 25 heads, but *at most* 25 heads. There could be 25, 24, 23, etc., down to 0. Although probability of getting any one of these counts — being far from average — should be low, there are also 26 of these events. Do the probabilities add up to very much? It turns out that the probability of getting 25 or fewer heads is tiny: about 2.818×10^{-7}.

The *scale* is very important in this discussion. If, out of 10 coin tosses, you got 4 or fewer heads, you shouldn't be too surprised. There is (roughly) a 37.7% chance of getting at most 4 heads. But if at most 40% of 1000 coin tosses came up heads, you should be *very* surprised. The odds of this occurring is occurring is roughly 1.364×10^{-10}.



The point generalizes to coins with any fixed probability of heads, $p \in [0,1]$. The *binomial distribution*, denoted $\mathcal{B}(n,p)$, is the distribution of the number of heads over n independent coin tosses that each flip heads with probability p. The probabilities of different binomial distributions is plotted above. (See also [Wikb].)

We write $B \sim \mathcal{B}(n,p)$ to denote a random variable $B \in \{0,\ldots,n\}$ drawn from the binomial distribution $\mathcal{B}(n,p)$. The expected value of B is $\mathbf{E}[B] = pn$. The following lemma bounds the probability of B being a multiplicative factor smaller than its mean, pn. Note that the probability decays *exponentially fast in the mean.*

Lemma 5.1. *Let $B \sim \mathcal{B}(n,p)$ and $\epsilon \in (0,1)$. Then*

$$\mathbf{P}[B \leq (1-\epsilon)pn] \leq e^{-\epsilon^2 pn/2}.$$

Proof. We have

$$\mathbf{P}[B \leq (1-\epsilon)pn] = \mathbf{P}\Big[e^{-\epsilon B} \geq e^{-\epsilon(1-\epsilon)pn}\Big] \overset{\text{(a)}}{\leq} \frac{\mathbf{E}\big[e^{-\epsilon B}\big]}{e^{-\epsilon(1-\epsilon)pn}} = e^{\epsilon(1-\epsilon)pn}\,\mathbf{E}\Big[e^{-\epsilon B}\Big] \tag{5.1}$$

by (a) Markov's inequality. It remains to analyze $\mathbf{E}\big[e^{\epsilon B}\big]$. Write B as the sum $B = X_1 + \cdots + X_n$, where each X_i is an independent $\{0,1\}$-random variable with $\mathbf{P}[X_i = 1] = p$. Now we have

$$\mathbf{E}\Big[e^{-\epsilon B}\Big] = \mathbf{E}\Big[e^{-\epsilon X_1 - \epsilon X_2 - \cdots - \epsilon X_n}\Big] \overset{\text{(b)}}{=} \mathbf{E}\Big[e^{-\epsilon X_1}\Big]\,\mathbf{E}\Big[e^{-\epsilon X_2}\Big]\cdots\mathbf{E}\Big[e^{-\epsilon X_n}\Big],$$

where (b) is because the X_i's are independent. For each X_i, we have

$$\begin{aligned}\mathbf{E}\Big[e^{-\epsilon X_i}\Big] &= pe^{-\epsilon} + (1-p) = 1 + p(e^{-\epsilon} - 1)\\ &\overset{\text{(c)}}{\leq} p\Big(1 - \epsilon + \epsilon^2/2\Big) + (1-p) = 1 - (\epsilon - \epsilon^2/2)p\\ &\overset{\text{(d)}}{\leq} e^{-(\epsilon-\epsilon^2/2)p}.\end{aligned}$$

Here (c) uses the inequality $e^{-x} \leq 1 - x + x^2/2$ for all $x > 0$[1]. (d) is by the inequality $1 + x \leq e^x$ for all x. Thus,

$$\mathbf{E}\Big[e^{-\epsilon B}\Big] = \mathbf{E}\Big[e^{-\epsilon X_1}\Big]\,\mathbf{E}\Big[e^{-\epsilon X_2}\Big]\cdots\mathbf{E}\Big[e^{-\epsilon X_n}\Big] \leq e^{-(\epsilon - \epsilon^2/2)pn},. \tag{5.2}$$

Putting everything together, we have

$$\mathbf{P}[B \leq (1-\epsilon)pn] \overset{\text{(e)}}{\leq} e^{\epsilon(1-\epsilon)pn}\,\mathbf{E}\Big[e^{-\epsilon B}\Big] \overset{\text{(f)}}{\leq} e^{\epsilon(1-\epsilon)pn - (\epsilon - \epsilon^2/2)pn} = e^{-\epsilon^2 pn/2},$$

by (e) inequality (5.1) and (f) inequality (5.2), as desired. □

One can prove a similar inequality bounding the probability that B exceeds its mean by a multiplicative factor. The proof is similar to Lemma 5.1 and left as exercise C.36.

Lemma 5.2. *Let $B \sim \mathcal{B}(n,p)$ and $\epsilon \in (0,1)$. Then for all $\mu \geq pn$*

$$\mathbf{P}[B \geq (1+\epsilon)\mu] \leq e^{-\epsilon^2 \mu/3}.$$

(The most common setting of Lemma 5.2 is when $\mu = pn$. But allowing $\mu \geq pn$ can sometimes be convenient.)

5.2 Random graphs

Paul Erdös, inspired by Ramsey [Ram30] before him, had a series of works analyzing *random graphs*. The results can mostly be grouped into two broad categories. First, he designed elaborate randomized constructions of graphs and showed that with nonzero probability, they can possess certain counterintuitive, seemingly impossible properties. This general approach is now called *Ramsey theory*. Second, he showed that for natural random graph models, these graphs – however random – tend to be extremely consistent about certain other properties.

This discussion falls in the latter category. We will study the $\mathcal{G}(n,p)$ random graph, sometimes called Erdös-Rényi graphs based on work by Erdös and Rényi [ER59; ER60]. A random graph from $\mathcal{G}(n,p)$ is an undirected graph over n vertices, where every edge is sampled independently with probability p. By now there is a large

[1]To see that

$$1 - x + x^2/2 \geq e^{-x}$$

for all $x \geq 0$, observe first that both sides equal 1 at $x = 0$. The derivative of the LHS, $-(1-x)$, is always at least the RHS of the derivative of the RHS, $-e^{-x}$, by the inequality $1 + y \leq e^y$ for all y.

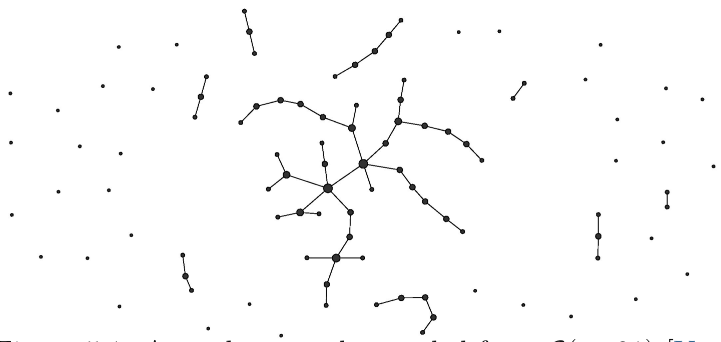


Figure 5.1: A random graph sampled from $\mathcal{G}(n, .01)$ [Von].

catalog of nontrivial and useful properties that, depending on p, are almost certain to appear or not appear in such a graph (for sufficiently large n). Moreover, Erdös and Rényi showed that these properties can vary dramatically with very small changes in p. Consider the following theorem.

Theorem 5.3. *Consider a random graph $G \sim \mathcal{G}(n, p)$ for $p = c/n$, where c is a constant.*

1. *If $c > 1$, then with high probability, there is exactly one connected component of G with $\Omega(n)$ vertices, and all other components have size $\leq O(\log n)$.*
2. *For $c < 1$, then with high probability, all connected components of G have size $< O(\log n)$.*

The parameter c above models the average degree (in expectation). The drama lies in the fact that a tiny change in the average degree c – from .999 to 1.0001 – flips the qualitative nature of a typical random graph from one of many tiny components to essentially one giant component. This is an example of a *threshold phenomena*; alternatively, a *nonlinear dynamic*. Such phenomena is not rare: it occurs in many situations in physics, as well as in models for epidemiology and social networks. Let us briefly mention — without claiming to be very precise — that the sensitivity to c gives some motivation for controlling the "reproductive number" when analyzing and preventing the spread of infectious diseases. The reproductive number is the expected number of healthy individuals that a sick individual effects.

We note that further research has obtained a much more refined and detailed understanding than stated in Theorem 5.3. We refer the reader to [Bol98, Chapter 7] for further details and other results in this area.

We present the proof for $c > 1$. The second case of $c < 1$ is left to the reader as exercise C.14.

5.2.1 Overview of the proof for $c > 1$

We will prove part 1 of Theorem 5.3 in roughly three parts.

Part 1: the gap theorem. Observe that in Theorem 5.3 above, regardless of the value of p, there are simply no “medium”-size components, such as a component of size $\sqrt{n}$ or of size $n/\log(n)$. The intermediate sizes are ruled out by the following “gap theorem”.

Lemma 5.4. *There is a universal constant $C > 0$, such that for all $\epsilon \in (0,1)$, and for all $n > 0$ sufficiently large, and $p = (1+\epsilon)/n$, we have the following. For a random graph $\mathcal{G}(n,p)$, with probability of error $\leq 1/n^2$, no component has k vertices for any value k in the interval*

$$\frac{C\log(n)}{\epsilon^2} \leq k \leq \frac{\epsilon n}{C}.$$

We analyze Lemma 5.4 theorem in Section 5.3. The proof makes a surprising connection to our discussion on random sums in Section 5.1.

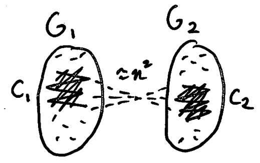
Part 2: existence of a large component. Lemma 5.4 establishes that all components are either very small or very big. However it does not assert that there are any big components. The next theorem, proven in Section 5.4 and based on analyzing a *Galton-Watson branching process*, shows that any single vertex has a reasonable chance of being in a component that is not small.

Lemma 5.5. *Let $p = (1+\epsilon)/n$ for $\epsilon > 0$. Let $v \in V$ be a vertex. For all $3 \leq h \leq \epsilon n$, with probability at least $1/h$, v has at least $1+h$ vertices in its connected component.*

Lemma 5.5 implies that there is almost certainly at least one giant component as follows. Let $h = c\log(n)/\epsilon^2$ for a sufficiently large constant c, and let $q = 1/h = \Omega(\epsilon^2/\log(n))$. Call a component “small” if it has at most h vertices. We want to argue that, for $p > 1/n$, there is at least one component that is not small. Since the gap theorem (Lemma 5.4) rules out all intermediate sizes, this would imply that there is at least one giant component of size $\Omega(\epsilon^2 n)$.

By Lemma 5.5, any vertex v has at least a probability q of not being in a small component. Now imagine a process where we first randomly select a vertex v and inspect its component. If it is not small, then we have obtained the non-small component we seek. Otherwise, if the component is small, then we throw out v and its component, and randomly select another vertex as v, and repeat. Each vertex we inspect has probability q of not being in a small component. We would have to fail on the order of n/h consecutive samples to conclude there is no small component - which happens with diminishingly small probability. Thus with very high probability, there is at least one component that is not small.

Part 3: uniqueness of the giant component. Can there be two giant components? The answer is no (with high probability) and here is a quick explanation. Instead of sampling from $\mathcal{G}(n,p)$ directly, we can first sample two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ from $G(n/2, p)$. In the second stage we can sample each cross-edge (v_1, v_2), where $v_1 \in V_1$ and $v_2 \in V_2$, independently with probability p. Now, by applying the theory we have already developed to G_1 and G_2, G_1 and G_2 will have some giant components, each of size $\Omega(\epsilon^2 n)$. Note that each graph can only have $O(1/\epsilon^2)$ of them.



Let C_1 be a giant component in G_1 and let C_2 be a giant component in G_2. We can sample up to $|C_1||C_2| \geq \Omega(\epsilon^4 n^2)$ edges between C_1 and C_2. Recalling that p is greater than $1/n$, the odds that all $\Omega(\epsilon^4 n^2)$ edges fail to be sampled is vanishingly small.[2] That is, we almost certainly connect C_1 and C_2. Since there are so few giant components, we will almost certainly connect all of them together. Thus, for $p > (1+\epsilon)/n$ for $\epsilon > 0$, we get a *unique* giant component. This establishes Theorem 5.3 for $c > 1$.

5.3 A gap in component size

In this section we prove Lemma 5.4, which asserts that when $p = (1+\epsilon)/n$ for a constant $\epsilon > 0$, then with high probability, all components are either very small or very large. Our analysis follows an approach due to Karp [Kar90]. His proof is also described in [Bol98]. We first restate Lemma 5.4 for the reader's convenience.

Lemma 5.4. *There is a universal constant $C > 0$, such that for all $\epsilon \in (0,1)$, and for all $n > 0$ sufficiently large, and $p = (1+\epsilon)/n$, we have the following. For a random graph $\mathcal{G}(n,p)$, with probability of error $\leq 1/n^2$, no component has k vertices for any value k in the interval*

$$\frac{C \log(n)}{\epsilon^2} \leq k \leq \frac{\epsilon n}{C}.$$

For a vertex $v \in V$, let $C(v) \subset V$ be the (randomized) component of v. To analyze $C(v)$, we imagine revealing $C(v)$ by a search algorithm. We maintain a collection

[2] We fail with probability $(1-p)^{\Omega(\epsilon^4 n^2)} \leq e^{-\Omega(\epsilon^4 n)}$.

of vertices known to be connected to v; initially just $\{v\}$. Each iteration i, starting from v, select a vertex v_i that is known to be in $C(v)$, but has not been explored. Then "explore" v_i by inspecting all of the edges incident to v_i, possibly adding to the collection of vertices known to be connected to v (but not yet explored).

We annotate this process as follows. For $i \in \mathbb{N}$, let

- v_i be the vertex that is explored in the ith iteration (or `nil` if all of $C(v)$ has already been explored).

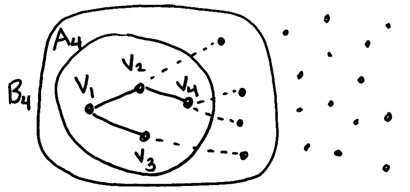
For each $i \in \mathbb{Z}_{\geq 0}$, let

- A_i be the set of vertices known to be in $C(v)$ after i iterations, and let
- B_i be the set of vertices that have been explored.

For the sake of concreteness, one can imagine processing the v_i's in BFS order. Recall that BFS marks each vertex when the vertex is first encountered, and if the vertex was unmarked, it is added to a queue. The next vertex visited is drawn from the queue. In terms of BFS, then, A_i is the set of vertices marked after i iterations, and B_i is the set of vertices that have left the queue and have been fully processed.

Ultimately, B_i, A_i, v_i are built up incrementally as follows.

1. Initially, we have $B_0 = \emptyset$ and $A_0 = \{v\}$.
2. In the first iteration, set $v_1 = v$, set $B_1 = \{v_1\}$, and set $A_1 = A_0 \cup N(v_1)$, where $N(v_1)$ is the (randomized) neighborhood of v_1.
3. In the ith iteration, if $B_{i-1} \neq A_{i-1}$, then select (any) $v_i \in A_{i-1} \setminus B_{i-1}$. Set $B_i = B_{i-1} \cup \{v_i\}$ and $A_i = A_{i-1} \cup N(v_i)$. Otherwise we terminate with $C(v) = B_{i-1} = A_{i-1}$.



The process terminates when $B_i = A_i$. But since $B_i \subseteq A_i$ and $|B_i| = i$, this is precisely when $|A_i| = i$. As long as $|A_i| > i$, A_{i+1} is generated by taking the union of A_i and a random sample of $V - A_i$ where each vertex is included with probability p. Thus we could have generated the sequence of A_i's instead by the following equivalent process, which omits any mention of B_i or v_i.

1. Initially set $A_0 = \{v\}$.
2. For each $i \in \mathbb{N}$, let S sample each vertex in $V \setminus A_{i-1}$ independently with probability p and set $A_i = A_{i-1} \cup S$.
3. Let i be the first index such that $|A_i| = i$, and return $C(v) = A_i$.

Fix an iteration i. The alternative (but equivalent) process described above exposes a simple distribution for A_i. For any vertex $x \neq v$, we have $x \notin A_i$ iff x failed to be added in each of the first i rounds, which occurs with probability exactly $(1-p)^i$. Moreover this event is independent across vertices. Thus $|A_i|$ is distributed exactly as the binomial distribution with $n-1$ coins and probability $1-(1-p)^i$; i.e., $|A_i| \sim \mathcal{B}(n-1, 1-(1-p)^i)$.

We first bound $\mathbf{E}[|A_i|]$.

Lemma 5.6. *Let $i \leq \epsilon n/2(1+\epsilon)$. Then $\mathbf{E}[|A_i|] \geq (1+\epsilon/2)i$.*

Proof. We have

$$(1-p)^i \leq e^{-ip} \leq 1 - ip + \frac{1}{2}(ip)^2 \overset{\text{(a)}}{\leq} 1 - ip + \epsilon ip/4 = 1 - (1-\epsilon/4)ip.$$

where (a) is because $ip = (1+\epsilon)i/n \leq \epsilon/2$. Thus

$$\mathbf{E}[|A_i|] = 1 + \left(1 - (1-p)^i\right)(n-1) \geq (1-\epsilon/4)ipn \overset{\text{(b)}}{\geq} (1+\epsilon/2)i.$$

For (b) we observe that $(1-\epsilon/4)(1+\epsilon) \geq (1+\epsilon/2)$ for $\epsilon > 0$ sufficiently small. □

Now we analyze the concentration of $|A_i|$ around its mean.

Lemma 5.7. *Let $i \leq \epsilon n/2(1+\epsilon)$. Then $\mathbf{P}[|A_i| \leq i] \leq e^{-\epsilon^2 i/8}$.*

Proof. We have

$$\mathbf{P}[|A_i| \leq i] \overset{\text{(a)}}{\leq} \mathbf{P}[|A_i| \leq (1-\epsilon/2)\,\mathbf{E}[|A_i|]] \overset{\text{(b)}}{\leq} e^{-\epsilon^2 i/8}.$$

Here (a) is by Lemma 5.6. (b) is by the tail inequality on binomial distributions, Lemma 5.1. □

To complete the proof of the gap theorem, let

$$I = \left\{ i \in \mathbb{N} : 32\ln(n)/\epsilon^2 \leq i \leq \epsilon n/2((1+\epsilon)) \right\}.$$

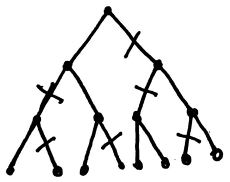


Figure 5.2: A complete binary tree of height 3, where each edge was deleted with probability 1/2.

For all $i \in I$, we have

$$\mathbf{P}[|A_i| \le i] \le 1/n^4.$$

By the union bound, we have

$$\mathbf{P}[|A_i| > i \text{ for all } i \in I] \ge 1 - \sum_{i \in I} \mathbf{P}[|A_i| \le i] \ge 1 - 1/n^3.$$

Thus with probability $\ge 1 - 1/n$, the number of vertices in the connected component of v, $|C(v)|$, does not lie in the range I. Taking the union bound over all $v \in V$ establishes part 1 of Lemma 5.4.

5.4 Galton-Watson process with general branching factors

We now move onto the second part of the analysis. By now we have established that there are (with high probability) no "medium" components – all component sizes have either at most $O(\log(n)/\epsilon^2)$ vertices, or at least $\Omega(\epsilon^2 n)$ vertices. Now we want to prove Lemma 5.5, which we first restate for the reader's convenience.

Lemma 5.5. *Let $p = (1+\epsilon)/n$ for $\epsilon > 0$. Let $v \in V$ be a vertex. For all $3 \le h \le \epsilon n$, with probability at least $1/h$, v has at least $1 + h$ vertices in its connected component.*

The proof is by relation to the so-called *Galton-Watson process* that arises in the study of reproducing populations. In the simplest case, imagine a population of size 1. Each generation, each member of the current generation flips k coins, each of which flips heads with probability $1/k$. For each heads, we generate another member of the next generation. The probabilities and number of coins are configured so that each member expects to have one child.

What is the probability that the population survives for h iterations, for a given parameter h? This is answered by the following.

Theorem 5.8. *Let T be a complete k-ary tree of height h, and suppose every edge is deleted independently with probability at most $1 - 1/k$. Then the probability that there is a leaf connected to the root is $\geq 1/h$ for $h \geq 3$, and $\geq (1 - e^{-1})^h$ for $h \leq 2$.*

An example of the case $k = 2$ is drawn in Fig. 5.2.

Proof. For $i \in \mathbb{N}$, let p_i be the probability that a particular node at height i is connected to a subleaf. We have $p_0 = 1$. For a node at height $i + 1$, the probability that there is no path to a leaf via a particular child is

$$1 - \frac{1}{k} + \frac{1}{k}(1 - p_i) = 1 - \frac{p_i}{k}.$$

By independence, we have

$$p_{i+1} = 1 - \left(1 - \frac{p_i}{k}\right)^k.$$

Observe that the RHS is increasing in p_i; thus to lower bound p_{i+1}, we can substitute any lower bound for p_i. We have

$$\begin{aligned}
&p_0 = 1, \\
&p_1 = 1 - (1 - 1/k)^k \geq 1 - e^{-1} \geq .63, \\
&p_2 = 1 - (1 - .63/k)^k \geq 1 - e^{-.63} \geq .467, \\
&p_3 \geq 1 - (1 - .467/k)^k \geq 1 - e^{-.467} \geq .373 \geq 1/3.
\end{aligned}$$

We claim by induction on i that $p_i \geq 1/i$ for all $i \geq 3$. The base case $i = 3$ was just proven. For the general case,

$$p_{i+1} \overset{(a)}{\geq} 1 - (1 - 1/ik)^k \geq 1 - e^{-1/i} \overset{(b)}{\geq} \frac{1}{i} - \frac{1}{2i^2} \geq \frac{1}{i+1}$$

Here (a) is by induction. (b) applies the inequality $1 + x \leq e^x$ for all x. (c) applies the inequality $e^x \leq 1 + x + \frac{1}{2}x^2$ for $x \leq 0$. □

5.4.1 Likelihood of small components

We can use the above branching process to analyze the probability that a given vertex v is in a component of size $\geq h$, for any $h \leq \epsilon n/(1 + \epsilon)$.

Fix $h \leq \epsilon n/(1 + \epsilon)$. Let $k = n/(1 + \epsilon)$. Consider a Galton-Watson process where each node produces up to k children, each with probability $1/k$. As proven above, the population lasts h generations with probability at least $1/h$.

Now, consider the following alternative randomized experiment: given the same Galton-Watson process, what is the probability that the populations produces a total of at least h individuals / nodes (including the root)? If the population survives h generations, then there are certainly at least h individuals. So the probability that the population produces at least h individuals is at least h.

We can imagine "revealing" the Galton-Watson outcome as follows. Initially we have the root node, without inspecting which of its k children "survive", and the root node is the only one known to survive. We select that node and reveal which of its k children survive. In general, as long as there is a surviving node where we have not sampled the children, we then sample the k children and reveal which of them survive. This experiment naturally terminates when we have revealed all the children of all the "survivors" and there is nothing left to sample. We also terminate early as soon as we identify at least h survivors.

This version explores the Galton-Watson tree node by node, and does not exploring the branches that die off. As a different pespective on the same phenomena, we have the same conclusion – the experiment produces at least h total (surviving) nodes with probability at least $1/h$.

We will relate this to our random graph as follows. Fix a starting vertex v; we want to analyze the probability that the connected component of v has at least h vertices. Initially, v is (obviously) in the same connected component as v, and we have not yet sampled of the neighbors of v. To explore the (randomized) connected component of v, we choose one vertex that we know is in the same component as v, but have not yet explored, and then reveal/sample all its neighbors. (The first chosen vertex is always v.) The process terminates when we have explored all the vertices known to be connected to w.

Consider the following adjustments to the process which is more conservative. First of all, we terminate immediately once we have identified h vertices connected to v. Second, suppose we have a vertex w that is known to be connected to v, but has not been explored. Since we have identified $< h$ vertices connected to v, there are at least $n - h \geq k$ vertices that are not yet (revealed to be) connected to v. Of these, we select k of them, sample those edges, and add the sampled neighbors to the collection of vertices connected to v.

This adjusted process is conservative relative to the real one – we omit sampling some edges, and we terminate as soon as we find h vertices connected to v. So the probability that the adjusted process finds h vertices connected to v is at most the probability that there are h vertices connected to v in the random graph. Moreover, the adjusted process maps directly to the adjusted k-ary Galton-Watson process mentioned above, where we are only interested in the probability that the population reaches h total individuals. Here v corresponds to the root, each vertex w connected to v corresponds to a node descending from v in the Galton-Watson tree, and "exploring" from w corresponds to sampling the children of w in the tree.

Putting everything, we conclude that for any $h \leq \epsilon n/(1+\epsilon)$, v is connected to at least h vertices with probability at least $1/h$. This gives us Lemma 5.5.

5.4.2 Directed graphs

One could naturally ask the same questions for directed graphs. Let $D(n,p)$ denote the distribution over *directed graphs* where every directed edge appears independently with probability p. We might similarly ask for the maximum number of vertices reachable from any component, or the size of the maximum strongly component.

It turns out that the analysis of directed graphs can be largely reduced to undirected graphs, as shown by Karp [Kar90] in the following delightfully simple way.

Theorem 5.9. *Let $G \sim G(n,p)$ and $D \sim G(n,p)$, and fix a vertex v. Then the size of the connected component of v in G, and the number of vertices reachable from v in D, are identically distributed.*

Proof. Let us introduce a second distribution of directed random graphs. Let $B(n,p)$ be the distribution of directed graphs where we sample each *undirected* edge $\{u,v\}$ independently with probability p, and for each sampled edge, add both directions (u,v) and (v,u) to the graph. Clearly, for a fixed vertex v, the size of v's (undirected) component in $G(n,p)$ is distributed identically to the number of vertices reachable from v in $B(n,p)$. We claim that the number of vertices reachable from v in $B(n,p)$ is identically distributed as in $D(n,p)$. At this point let us simply quote Karp [Kar90, Lemma 1] (with minor changes in notation) whose proof is very elegant.

> ...To see that the last two random variables are identically distributed, note that the probability spaces $B(n,p)$ and $D(n,p)$ differ in only one respect: a digraph G drawn from $B(n,p)$, are (u,v) is present if and only if arc (v,u) is present, while, in a digraph D drawn from $D(n,p)$, then the event that (v,u) is present is independent of the event that (u,v) is present. Thus no experiment based on checking for the presence or absence of arcs can distinguish between the two probability spaces unless it checks both an arc and its reversal. But any standard sequential algorithm, such as breadth-first search or depth-first search, for building a search tree containing exactly the vertices reachable from vertex 1, checks for the presence of arc (u,v) only if vertex u is in the search tree and v is not;

thus it never checks both an arc and its reversal, and accordingly cannot distinguish $B(n,p)$ from $D(n,p)$.

To summarize the excerpt, standard search algorithms for reachability do not distinguish $B(n,p)$ and $D(n,p)$ anyway, so the number of reachable vertices is identically distributed. □

5.5 Additional notes and materials

Lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

5.6 Exercises

Exercise 5.1. Prove Item 2 of Theorem 5.3.[3]

Exercise 5.2. Prove Lemma 5.2. (Here the important part is not the constant, $1/3$ – any constant $c > 0$ is already interesting.)

[3]It may be helpful to understand the proof of the gap theorem (Lemma 5.4) in Section 5.3.

Chapter 6

Randomized Rounding

Many of the problems we are interested in are inherently discrete, and unfortunately many discrete problems are NP-Hard. One general class of NP-hard problems are *integer programs* (IPs), which are linear programs where the variables are required to be integers.

One example is a *covering integer program*, which is an optimization problem of the form

$$\begin{aligned} \text{minimize } & \sum_{j=1}^{n} c_j x_j \\ \text{over } & x \in \mathbb{Z}_{\geq 0} \\ \text{s.t. } & \sum_{j=1}^{n} A_{ij} x_j \geq b_i \text{ for } i \in [m], \end{aligned} \tag{6.1}$$

where $A \in \mathbb{R}_{\geq 0}^{m \times n}$, $b \in \mathbb{R}_{>0}^{m}$, and $c \in \mathbb{R}_{>0}^{n}$. Here, each variable $x_j \in \mathbb{Z}_{\geq 0}$ models a binary decision. Each c_j can be interpreted as the *cost* of taking $x_j = 1$. For $i \in [m]$, the constraint

$$\sum_{j=1}^{n} A_{ij} x_j \geq b_i \text{ for } i \in [m]$$

is a *covering constraint*, saying we must set $x_j = 1$ for enough variables x_j so that the sum of A_{ij} over these variables is at least b_i.

In the set cover problem, we have a family of n sets $\mathcal{F} = \{S_1, \dots, S_n\}$, where each set S_i is a subset of m points $\mathcal{U}$. The goal is to compute the minimum cardinality collection of sets, $S_{i_1}, \dots, S_{i_k}$ where $i_1, \dots, i_k \in [n]$, such that

$$S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_k} = \mathcal{U}.$$

The set cover problem is NP-Hard.

Set cover is the special case of a CIP where $c = \mathbb{1}$, $b = \mathbb{1}$, and $A \in \{0,1\}^{m \times n}$. The coordinates $j \in [n]$ correspond to sets, and the coordinates $i \in [m]$ correspond to points. Set j covers point i iff $A_{ij} = 1$.

Besides covering integer programs, there are also packing integer programs, which are maximization problems subject to packing (i.e., $\leq$) constraints, of the form

$$\begin{aligned} \text{maximize } & \sum_{j=1}^{n} b_j y_j \\ \text{over } & y \in \mathbb{Z}_{\geq 0}^{n} \\ \text{s.t. } & \sum_{j=1}^{m} A_{ij} y_j \leq c_j \text{ for } i \in [m] \end{aligned}$$

where $A \in \mathbb{R}_{\geq 0}^{m \times n}$, $b \in \mathbb{R}_{>0}^{n}$, and $c \in \mathbb{R}_{>0}^{m}$. Here each b_j can be understood as the "profit" of taking $y_j = 1$, and we want to maximize the total profit. We are constrained by the n packing constraints. The knapsack problem is a special case of the covering integer programs where $m = 1$.

In general, integer programs maximize or minimize a set of integer variables over a linear objective, subject to linear equality and inequality constraints.[1] The basic appeal of integer programs is that they are very flexible for modeling discrete optimization problems. Unfortunately, their broad applicability also makes them NP-Hard.

Integer programs are NP-Hard because the output is required to be discrete. If we allowed the variables to vary *continuously* over the reals, we instead have a *linear program* (LP). For example, the following describes a linear program for covering problems similar to the covering integer program above. This time, however, each variable x_j is allowed to take any nonnegative value.

$$\begin{aligned} \text{minimize } & \sum_{j=1}^{n} c_j x_j \\ \text{over } & x \in \mathbb{R}_{\geq 0}^{n} \\ \text{s.t. } & \sum_{j=1}^{n} A_{ij} x_j \geq b_i \text{ for } i \in [m]. \end{aligned}$$

Note that any solution feasible to the CIP (6.1) is also feasible for the LP above. As such, the LP is said to be a *relaxation* of the CIP. It implies that the optimum value of the LP is less than or equal to the optimum value for the integer program.

[1]Integer programs may have both packing (i.e., $Ax \leq b$) and covering (i.e., $Ax \geq b$) constraints. "Packing integer programs" and "covering integer programs" refer to the special case that have only packing constraints or only covering constraints, respectively.

Unlike integer programs, linear programs are polynomial time solvable (!). This allows for the following general approach to discrete optimization: given an IP formulation of the problem, instead solve the corresponding LP. The LP provides a *fractional solution* satisfying the same constraints, which we can treat as a clue towards a good integer solution. The goal becomes to convert the fractional solution to an integer solution, while maintaining feasibility and the objective value. There are several strategies to *round* a fractional solution to an integer solution and they are covered in courses on approximation algorithms (e.g., [WS11; Vaz01]). We will study a technique called *randomized rounding*.

In this chapter, we will use randomized rounding to obtain approximation algorithms for max SAT (Section 6.1), set cover (Section 6.2), and CIPs (Section 6.3).

6.1 SAT

Recall the max-SAT problem from Chapter 1: given a boolean formula $f(x_1, \ldots, x_n)$ in CNF, the goal is to find an assignment of $x_1, \ldots, x_n$ to $\{\mathtt{t}, \mathtt{f}\}$ that satisfies the maximum number of clauses. There we showed that a random assignment gives a 7/8-approximation for 3-SAT. Moreover the algorithm can be derandomized, and the approximation factor is best possible unless $P = NP$.

More generally a random assignment gives a $(1 - 2^{-k})$ approximation for k-SAT, for any $k \in \mathbb{N}$. The following table lists the approximation factors for the first few values of k.

k	**oblivious APX**
1	1/2
2	3/4
3	7/8
4	15/16
5	31/32
$\vdots$	
k	$1 - 1/2^k$

The approximation factor gets better and better as k increases. Meanwhile, the first row — $k = 1$ — is rather embarrassing, since 1-SAT is trivial. 2-SAT is not as trivial but there is a polynomial time algorithm for this problem as well. So oblivious rounding is not so great for very small values of k.

This is relevant as we now consider the more general form of max-SAT, where we are given a formula in CNF where each clause can have any number of clauses. We can still apply our oblivious randomized algorithm that flips a coin for every variable.

If every clause had *at least* k variables, then we obtain a $1 - 1/2^k$ approximation ratio. But the presence of single-variable clauses – which one might expect to cause the least trouble — means we only have a 1/2-approximation ratio in general. We will use linear programming to improve the approximation ratio.

An integer program for SAT. We first translate max-SAT to an integer program. Fix a formula $f(x_1, \dots, x_n)$ of n variables, consisting of m clauses $C_1, \dots, C_m$. Let OPT denote the maximum number of satisfiable clauses. Let "$x_j \in C_i$" indicate that the symbol x_j appears in C_i (without negation) and "$\bar{x}_j \in C_i$" indicate that $\bar{x}_j$ appears in C_i.

Consider now the following integer program.

$$\begin{aligned} \text{maximize } & \sum_{i=1}^m z_i \text{ over } y \in \{0,1\}^n,\, z \in \{0,1\}^m \\ \text{s.t. } & \sum_{j : x_j \in C_i} y_j + \sum_{j : \bar{x}_j \in C_i} (1 - y_j) \geq z_i \text{ for all clauses } C_i. \end{aligned}$$

The integer program has $\{0,1\}$-variables for each variable x_j and each clause C_i, with the following interpretations.

1. For $j = 1, \dots, n$, let $y_j \in \{0,1\}$ indicates whether we set $x_j = \mathtt{t}$ ($y_j = 1$) or $x_j = \mathtt{f}$ ($y_j = 0$).
2. For $i = 1, \dots, m$, let $z_i \in \{0,1\}$ indicate whether we satisfy the ith clause ($z_i = 1$) or not ($z_i = 0$).

The integer program seeks to maximize the number of satisfied clauses, represented by $\sum_{i=1}^m z_i$. For each clause C_i, the corresponding constraint implies we can only set $z_i = 1$ if $y_j = 1$ for some $x_j \in C_i$ or $y_j = 0$ for some $\bar{x}_j \in C_i$.

A linear program for SAT. As discussed, while we cannot solve integer programs in polynomial time, we can solve linear programs. We relax the integer program above to a linear program by now allowing each variable y_j or z_i to lie anywhere in the interval $[0,1]$.

$$\begin{aligned} \text{maximize } & \sum_{i=1}^m z_i \text{ over } y_1, \dots, y_n, z_1, \dots, y_m \in \mathbb{R} \\ \text{s.t. } & \sum_{j : x_j \in C_i} y_j + \sum_{j : \bar{x}_j \in C_i} (1 - y_j) \geq z_i \text{ for all clauses } C_i \\ & 0 \leq z_i \leq 1 \text{ for all } i = 1, \dots, m \\ & 0 \leq y_j \leq 1 \text{ for all } j = 1, \dots, n \end{aligned}$$

Let OPT_{LP} denote the optimum value of the LP. Since the LP is a relaxation of the IP above it, we have $\text{OPT}_{\text{LP}} \geq \text{OPT}$.

Rounding the LP solution. Let $y_1, \dots, y_n$ and $z_1, \dots, z_m$ be an optimum solution to the LP. We know that the objective value $\text{OPT}_{\text{LP}} = \sum_i z_i$ is very good. Our goal now is to convert the y_j's into *discrete* decisions while keeping the objective value as close to OPT_{LP} as possible.

The basic question is: how do we interpret a *fractional* value such as $y_1 = .5$? The LP seems to suggest that we should set x_1 to be one-half true and one-half false. Of course the are no half values in boolean algebra and "one-half true and one-half false" is total nonsense. A different interpretation is that of a randomized experiment where we set $x_1 = \texttt{t}$ *half the time*, and $x_1 = \texttt{f}$ the other half. Consider the following *randomized rounding* algorithm:

```
Randomized rounding for SAT
1. Let $y_1, \dots, y_n$ be an optimum solution to the LP for max-SAT.
2. For each variable $x_j$, independently, randomly set $x_j = \texttt{t}$ with
   probability $y_j$, and $x_j = \texttt{f}$ otherwise.
```

Next we analyze the expected number of clauses satisfied by randomized rounding. By linearity of expectation this boils down to analyzing the probability of satisfying each individual clause.

Lemma 6.1. *Each clause C_i is satisfied with probability at least $(1 - 1/e)z_i$.*

Proof. A clause C_i is not satisfied iff we randomly set $x_j = \texttt{f}$ for all $x_j \in C_i$, and set $x_j = \texttt{t}$ for all $\bar{x}_j \in C_i$. Thus

$$\mathbf{P}[C_i \text{ not satisfied}] = \prod_{j : x_j \in C_i} (1 - y_j) \prod_{j : \bar{x}_j \in C_i} y_j.$$

Now, by the inequality $1 + x \leq e^x$ (for all x), we can simply the RHS as

$$e^{-\sum_{j : x_j \in C_i} y_j + \sum_{j : \bar{x}_k \in C_i} (1 - y_j)} \leq e^{-z_i}.$$

Here the inequality follows from the LP constraint for C_i. Finally, by convexity[2] of $f(x) = e^{-x}$, we have

$$e^{-z_i} \leq (1 - z_i)e^0 + z_i e^{-1} = 1 - (1 - 1/e)z_i,$$

as desired. ☐

[2] $f(x)$ is convex if $f(ta + (1-t)b) \leq tf(a) + (1-t)f(b)$ for all a, b and all $t \in [0, 1]$.

By linearity of expectation, the expected number of clauses that are satisfied equals the sum of probabilities of each clause being satisfied. Therefore, by Lemma 6.1, we will satisfy at least $(1 - 1/e)\,\mathrm{OPT}_{\mathrm{LP}} \geq (1 - 1/e)\,\mathrm{OPT}$ clauses in expectation.

Theorem 6.2. *There is a $(1 - 1/e)$-approximation algorithm for max-SAT.*

Note that our bound is only interesting when there are clauses with one or two variables; otherwise oblivious rounding is still better.

The best of both worlds. Part of the problem is that Lemma 6.1 is not tight for small k. Take for example $k = 1$. Obviously, if $|C_i| = 1$, then C_i is satisfied with probability z_i, not $(1 - 1/e)z_i$. For $k = 2$, we have the following better analysis.

Lemma 6.3. *If $|C_i| = 2$, then C_i is satisfied with probability $\geq 3z_i/4$.*

Proof. Suppose for simplicity that $C_i = x_1 \vee x_2$. (It will be obvious how to generalize the analysis to other pairs of variables.) We have

$$\begin{aligned}
\mathbf{P}[C_i \text{ not satisfied}] = (1-y_1)(1-y_2) &\overset{(a)}{\leq} \left(\frac{(1-y_1)+(1-y_2)}{2}\right)^2 \\
&\leq \left(1 - \frac{z_j}{2}\right)^2 \overset{(b)}{\leq} (1-z_j)\left(1 - \frac{0}{2}\right)^2 + z_j\left(1 - \frac{1}{2}\right)^2 \\
&= 1 - z_j + \frac{z_j}{4} = 1 - \frac{3}{4}z_j.
\end{aligned}$$

(a) is by AM-GM. (b) is by convexity of $f(x) = \left(1 - \frac{x}{2}\right)^2$. □

Below we list the probability of clause being satisfied by the oblivious and LP rounding strategies, as a function of the number of variables in the clause, k. Observe that the average of the probabilities is at least $(3/4)z_i$ for all k, as indicated in the column on the right.

k	**oblivious**	**LP**	**average**
1	1/2	z_i	$\geq (3/4)z_i$
2	3/4	$(3/4)z_i$	$\geq (3/4)z_i$
3	7/8	$(1-1/e)z_i$	$\geq (3/4)z_i$
4	15/16	$(1-1/e)z_i$	$\geq (3/4)z_i$
5	31/32	$(1-1/e)z_i$	$\geq (3/4)z_i$
$\vdots$			
k	$1 - 1/2^k$	$(1-1/e)z_i$	$\geq (3/4)z_i$.

The table suggests that we might be able to merge the oblivious sampling and LP rounding algorithms to obtain a 3/4-approximation ratio. But the two approaches seem like polar opposites: one approach is completely oblivious, and the other strongly depends on the formula (implicitly via the LP solver). But here's a trick: just pick one of the two strategies uniformly at random. The "averaging" will work itself out in the analysis.

Hybrid algorithm for max-SAT.

1. With probability 1/2, return a uniformly random assignment.
2. Otherwise solve and randomly round the LP.

Theorem 6.4. *The hybrid algorithm gives a* $(3/4)$*-approximation algorithm for max-SAT.*

We leave the analysis to the reader as exercise C.40.

6.2 Set cover

In *set cover*, we are given m points $[m] = \{1, \dots, m\}$, and n sets $S_1, \dots, S_n \subseteq [m]$. The goal is to

find the minimum number of sets $S_{i_1}, \dots, S_{i_k}$ *such that* $S_{i_1} \cup \cdots \cup S_{i_k} = [m]$.

Some natural extensions including adding costs for sets, pointwise demands that require points to be covered by multiple sets, and coefficients $A_{ij} \in [0,1]$ that indicate the amount of coverage a set S_i gives to point j. These will be considered when we discuss CIPs later. Set Cover is NP-Hard. Instead we will design an approximation algorithm for set cover, via randomized rounding.

We first write an LP relaxation for set cover. For each set S_j we introduce a variable x_j that models our decision to take S_j in our set cover. Consider the following LP.

$$\text{minimize } \sum_{j=1}^{n} x_j \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } \sum_{S_j \ni i} x_j \geq 1 \text{ for all } i \in [m].$$

The objective, $\sum_{j=1}^{n} x_j$, is the (fractional) number of sets in our (fractional) set cover. For each point $i \in [m]$, we require at least one fractional set among the family of sets that cover that point.

Randomized rounding. Let us consider the following randomized rounding algorithm

```
1. Let $x_1, \ldots, x_n$ be an optimum solution to the set cover LP.
2. For each set $S_j$, independently, take $S_j$ with probability $x_j$.
```

Let $F \subseteq \{S_1, \ldots, S_m\}$ be the random family of sets produced by randomized rounding. We have two questions to address:

1. How big is F, relative to OPT?
2. Is F a set cover?

For the first question, we have

$$\mathbf{E}[|F|] = \sum_j \mathbf{P}[S_j \in F] = \sum_j x_j = \mathrm{OPT}_{\text{LP}},$$

which is very good indeed. Now, is F a set cover? Fix a point $i \in [m]$. The expected number of sets covering i is

$$\mathbf{E}[\text{\# sets in } F \text{ covering } i] = \sum_{S_j \ni i} \mathbf{P}[S_j \in F] = \sum_{S_j \ni i} x_j \geq 1.$$

So i expects to have at least 1 set in F containing it. But this does not imply that F is a set cover — that F covers all of $[m]$ simultaneously — with any nonnegligible probability. A better question is about the probability that F covers i, or rather, the probability that F doesn't cover i. F does not cover i iff it fails to sample any of the sets covering i, hence

$$\mathbf{P}[F \textit{ doesn't} \text{ cover } i] = \prod_{S_j \ni i} (1 - x_j) \leq e^{-\sum_{S_j \ni i} x_j} \leq e^{-1}.$$

So each point is covered with constant probability.

The question is: how do we increase the probability that *all points are covered*, simultaneously?

Scaling and rounding. The problem above is that having each point covered with constant probability does *not* imply that all points are covered with constant, or any nonnegligible, probability. However, having each point covered with *high probability* does imply, *by the union bound*, that all points are covered with *high probability*. Now, how can we ensure that each point is covered with high probability? By scaling up x before rounding.

Randomized rounding for set cover.

1. Let $x_1, \ldots, x_n$ be an optimum solution to the set cover LP. *Let* $\alpha = 2\log(m)$.
2. For each set S_j independently, take S_j with probability $\min\{1, \alpha x_j\}$.

As before, let F denote the random collection of sets returned by the algorithm.

Lemma 6.5. *Each point i is covered with probability* $1 - 1/m^2$.

Proof. If $x_j \geq 1/\alpha$ for any set S_j covering i, then i is covered deterministically. Otherwise, by similar calculations as before, we have

$$\mathbf{P}[F \text{ doesn't cover } i] = \prod_{S_j \ni i} (1 - \alpha x_j) \leq e^{-\sum_{S_j \ni i} \alpha x_j} \leq e^{-\alpha} = 1/m^2. \tag{6.2}$$

□

Lemma 6.6. *F is a set cover with probability at least* $1 - 1/m$.

Proof. By the union bound, we have

$$\mathbf{P}[F \text{ is not a set cover}] \leq \sum_i \mathbf{P}[F \text{ doesn't cover } i] \leq \frac{1}{m},$$

as desired. □

Theorem 6.7. *Randomized rounding is a $O(\log m)$-approximation algorithm for set cover.*

Proof. We have $\mathbf{E}[|F|] = 2\log(m)\,\mathrm{OPT}$, and by Markov's inequality, $|F| \leq 4\log(m)\,\mathrm{OPT}$ with probability $1/2$. F is also a set cover with probability $1 - 1/m$. By the union bound, with probability of error at most $1/2 + 1/m$, F is a set cover of size at most $4\log(m)\,\mathrm{OPT}$. □

6.3 Covering integer programs

We consider covering integer programs. They are discrete problems generalizing set cover, of the form

$$\text{minimize } \langle c, x \rangle \text{ over } x \in \mathbb{Z}^n_{\geq 0} \text{ s.t. } Ax \geq b.$$

where $c \in \mathbb{R}^n_{>0}$, $A \in \mathbb{R}^{m \times n}_{\geq 0}$, and $b \in \mathbb{R}^m_{>0}$. The matrix notation expands out to the following.

$$\begin{aligned} \text{minimize } & \sum_{j=1}^n c_j x_j \\ \text{over } & x \in \mathbb{Z}^n_{\geq 0} \\ \text{s.t. } & \sum_{j=1}^n A_{ij} x_j \geq b_i \text{ for } i \in [m]. \end{aligned}$$

We assume without loss of generality that $A_{ij} \leq b_i$ for all i. (Why?) We also assume without loss of generality that m is at least some constant; say, 4. (Otherwise add a few empty constraints.)

The algorithm we analyze is essentially the same as for set cover. We scale up x by a $O(\log m)$ factor, and then randomly round each coordinate of x, independently, to integer values.

1. Let $\alpha = 8\log(m)$.
2. For each $j \in [n]$, independently, let

$$z_j = \begin{cases} \lceil \alpha x_j \rceil & \text{with probability } \alpha x_j - \lfloor \alpha x_j \rfloor, \\ \lfloor \alpha x_j \rfloor & \text{otherwise.} \end{cases}$$

The algorithm is very similar to set cover. However, the coefficients A_{ij} make the analysis more difficult. For set cover, we were able to give an exact formula for the probability that a point i is uncovered. (Equation (6.2).) Here it is not so simple, and we instead appeal to the *multiplicative* (or *relative*) Chernoff bound introduced last chapter. The proof is deferred to Section 6.A.

Theorem 6.8. *Let $X_1, \dots, X_n \in [0,1]$ be independent random variables, and $\epsilon \in [0,1]$.*

- *For $\mu \geq \mathbf{E}[X_1 + \cdots + X_n]$,*
$$\mathbf{P}[X_1 + \cdots + X_n \geq (1+\epsilon)\mu] \leq e^{-\epsilon^2 \mu/3}.$$
- *For $\mu \leq \mathbf{E}[X_1 + \cdots + X_n]$,*
$$\mathbf{P}[X_1 + \cdots + X_n \leq (1-\epsilon)\mu] \leq e^{-\epsilon^2 \mu/2}.$$

The main part of the analysis is to show that a single constraint is satisfied with high probability, as follows.

Lemma 6.9. *The ith constraint is satisfied with probability at least* $1 - 1/m^2$.

Proof. By scaling, we may assume that $b_i \leq 1$, hence $A_{ij} \leq 1$ for all j. We may assume that $\sum_j A_{ij}\lfloor \alpha x_j \rfloor < 1$, since otherwise the constraint is satisfied deterministically.

For each coordinate j, let $X_j = A_{ij}(z_j - \lfloor \alpha x_j \rfloor)$ for each j. Let $\mu = \mathbf{E}\Big[\sum_{j=1}^n X_j\Big]$. We have

$$\mu = \sum_{j=1}^n A_{ij}(\alpha x_j - \lfloor \alpha x_j \rfloor) \geq \alpha - \sum_{j=1}^n A_{ij}\lfloor \alpha x_j \rfloor.$$

This implies that both

$$\mu \geq \alpha\left(1 - \sum_{j=1}^n A_{ij}\lfloor \alpha x_j \rfloor\right), \text{ and } \mu \geq \alpha - 1 \geq 8\log(m).$$

We have

$$\begin{aligned}\mathbf{P}[z \text{ fails constraint } i\ ] &= \mathbf{P}\left[\sum_{j=1}^n A_{ij} z_j \leq 1\right] \\ &= \mathbf{P}\left[\sum_{j=1}^n X_j \leq 1 - \sum_{j=1}^n A_{ij}\lfloor \alpha x_j \rfloor\right] \\ &\leq \mathbf{P}\left[\sum_{j=1}^n X_j \leq \frac{\mu}{\alpha}\right].\end{aligned}$$

By the Chernoff inequality, we have

$$\mathbf{P}\left[\sum_{j=1}^n X_j \leq \frac{\mu}{\alpha}\right] \leq e^{-(1-1/\alpha)^2\mu/2}.$$

Now,

$$\frac{1}{2}\left(1 - \frac{1}{\alpha}\right)^2 \mu \geq \frac{1}{2}\left(1 - \frac{1}{\alpha}\right)^3 \alpha \geq \frac{\alpha}{4} \geq 2\log(m).$$

Thus

$$\mathbf{P}[z \text{ fails constraint } i] \leq e^{-(1-1/\alpha)^2\mu/2} \leq e^{-2\ln(m)} = \frac{1}{m^2},$$

as desired. □

Theorem 6.10. *With constant probablity, the randomized rounding algorithm returns a feasible solution of cost at most* $O(\ln(m)) \operatorname{OPT}_{\text{LP}}$.

Proof. By Markov's inequality, we have

$$\sum_j c_j z_j \leq 2\,\mathbf{E}\left[\sum_j c_j z_j\right] = 2\alpha \operatorname{OPT}_{\text{LP}}$$

with probability of error at most $1/2$. Each constraint i is satisfied with probability of error at most $1/m^2$.

By the union bound, the probability of z not being a feasible solution of cost at most $2 \operatorname{OPT}_{\text{LP}}$ is at most

$$\frac{1}{2} + \mathbf{P}[z \text{ fails any constraint}] \leq \frac{1}{2} + \sum_i \mathbf{P}[z \text{ fails constraint } i] \leq \frac{1}{2} + \frac{1}{m},$$

as desired. □

6.4 Additional notes and materials

Lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2025 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2023 lecture materials. Click on the links below for the following files:

- Handwritten .pdf prepared before the lecture (and .note file).
- Handwritten .pdf annotated during the presentation (and .note file).
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

6.5 Exercises

Exercise 6.1. Complete the proof of Theorem 6.4.

Exercise 6.2. Extend the approximation algorithm for set cover to positive costs. For each set, there is a positive cost $c_j > 0$. The goal is to compute the minimum cost collection of sets that covers all the points.

Exercise 6.3. Design and analyze a deterministic 3/4-approximation algorithm for max-SAT.[3]

Exercise 6.4. Consider an instance of (weighted) set cover defined by sets $S_1, \dots, S_n \subseteq [m]$ and costs $c_i > 0$ for each set S_i. The goal is to compute the minimum cost collection of sets covering $[m]$. We saw that solving the LP and then randomly rounding gives a $O(\log m)$ approximation. Here we consider a special case where all the sets are small and obtain a better approximation factor by a standard extension of randomized rounding called *alterations*.

Let $\Delta \in \mathbb{N}$ be such that $|S_j| \leq \Delta$ for all j. Consider the algorithm `round-and-fix` for which some speudocode is given below. `round-and-fix` is similar to randomized rounding and has two stages. The first stage solves the LP and then *rounds* the solution scaled up by some factor $\alpha \geq 1$. It is possible that some of the elements $i \in [m]$ may not be covered. In the second stage, we *fix* each uncovered element by (deterministically) taking the cheapest set that covers it.

[3]You may first want to design and analyze a deterministic $(1 - 1/e)$-approximation algorithm for max-SAT.

```
round-and-fix(sets $S_1, \ldots, S_n \subseteq [m]$, costs $c \in \mathbb{R}^n_{>0}$, $\alpha \geq 1$)
1. let $x \in [0,1]^n$ solve the set cover LP
2. let $F \subseteq \{S_1, \ldots, S_n\}$ sample each set $S_i$ independently with
   probability $\min\{1, \alpha x_i\}$
3. for each $i \in [m]$
   A. if $i$ is not covered by $F$
      1. add the cheapest set covering $i$ to $F$
4. return $F$
```

Show that for an appropriate choice of α, this algorithm returns a $O(\log \Delta)$ approximation to the set cover instance (in expectation). (It is possible to get $\log \Delta + \log\log \Delta + O(1)$ with care.)

Exercise 6.5. The defining characteristic of LPs is that the objective and all linear constraints are given by linear functions. It is natural to generalize this notion and consider mathematical programs where the objective and linear constraints are all given by low-degree polynomials; say, bounded by a degree d. Let us call these "degree d polynomial programs". Linear programs are degree 1 polynomial programs.

Prove that degree d polynomial programs are NP-Hard to solve for $d \geq 3$. To this end, pick a suitable NP-Hard problem, and design a degree 3 polynomial program that can be rounded to a discrete solution without any loss.[4]

Exercise 6.6. For CIPs, we could assume without loss of generality that $A_{ij} \leq b_j$ for all i, j. (In fact this assumption was critical for applying the Chernoff bound.) Suppose now that we had $\lambda A_{ij} \leq b_j$ for all i, j for a parameter $1 \leq \lambda \leq \log(n)$. Design and analyzing a $O(\log(m)/\lambda)$-approximation algorithm for this setting.

Exercise 6.7. In CIPs we allowed each x_j to be as large as we want. Suppose we added the constraint $x_j \leq 1$ for all j. Would the randomized rounding algorithm from Section 6.3 still obtain a $(1 - 1/e)$-approximation ratio? Why or why not?

Exercise 6.8. Recall the set cover problem for which we obtained a randomized $O(\log n)$-approximation. Here we consider a (maximum weight) set *packing* problem, defined as follows.

[4]Better yet, prove the same for $d \geq 2$.

Let $[m]$ be a set of points, and let $S_1, \ldots, S_n \subseteq [m]$ be n subsets of $[m]$. Let $b_1, \ldots, b_n > 0$ represent the profit of $S_1, \ldots, S_n$, respectively. We say that a collection of sets $\mathcal{F} = \{S_{j_1}, \ldots, S_{j_k}\}$ is a *set packing* if they are all disjoint. The total profit of such a set packing is defined as the sum of profits $b_{j_1} + \cdots + b_{j_k}$ of the corresponding sets.

The goal is to compute a set packing of maximum profit, but the problem is NP-Hard. Here we consider the following (perhaps unusual) approximation criteria. Let OPT denote the maximum profit of any set packing. For $\alpha \geq 1$, we say that a collection of sets $S_{j_1} + \cdots + S_{j_k}$ is an α-packing if each point is covered by at most α sets S_{j_h}. We say that a randomized collection of sets $\mathcal{F}$ is a *randomized approximate α-packing* if

1. The expected total profit of $\mathcal{F}$ is at least OPT.
2. With high probability, $\mathcal{F}$ is an α-packing.

Design and analyze a polynomial time algorithm that outputs a randomized approximate α-packing for α as small as possible.[5]

Exercise 6.9. Our department has a large catalog of classes and a handful of course requirements you have to satisfy to graduate. The requirements seem to be particularly complicated for undergraduates. Each requirement is defined by a subset of classes and a number specifying the number of courses you have to take from this subset.

Formally, let the courses be indexed 1 through n. We have L requirements $(S_1, k_1), \ldots, (S_L, k_L)$, each consisting of a set $S_i \subseteq [n]$ and an integer $k_i \in \mathbb{N}$. For each i, you have to take at least k_i classes from S_i. Let $T \subseteq [n]$ be the set of classes you've already taken. Let $m = \sum_i |S_i|$ denote the total size of all the sets.

Now we have two possible interpretations of the rules. In the first version, a single class can only count towards a single requirement. In the second version, a single class can count towards any number of requirements. For example, suppose you are required to take $(k_1 = 2)$ classes from $S_1 = \{A, B, C\}$ and $(k_2 = 2)$ classes from $S_2 = \{C, D, E\}$, and you have already taken $T = \{B, C, D\}$. In the first version, you do not have enough classes to graduate; in the second version, you do have enough classes to graduate.

Here we have related but slightly different problems for the two systems.

[5]Here we are interested in the approximation factor α – the smaller and closer to 1 the better – and not the exact polynomial running time.

1. (5 points) Suppose a single class can only count towards a single requirement. Consider the problem of deciding if you have already taken enough classes to graduate. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

2. (5 points) Now suppose a single class can count towards any number of requirements. Consider the problem of identifying the minimum number of additional classes that need to be taken to satisfy all the requirements. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

6.A Proof of the multiplicative Chernoff bound

Theorem 6.8. *Let $X_1, \dots, X_n \in [0,1]$ be independent random variables, and $\epsilon \in [0,1]$.*

- *For $\mu \geq \mathbf{E}[X_1 + \cdots + X_n]$,*
$$\mathbf{P}[X_1 + \cdots + X_n \geq (1+\epsilon)\mu] \leq e^{-\epsilon^2 \mu / 3}.$$
- *For $\mu \leq \mathbf{E}[X_1 + \cdots + X_n]$,*
$$\mathbf{P}[X_1 + \cdots + X_n \leq (1-\epsilon)\mu] \leq e^{-\epsilon^2 \mu / 2}.$$

Proof. We prove the claim for a slightly weaker constant in the exponent. Let $t \in [0,1]$ be a parameter TBD. We have

$$\mathbf{P}[X_1 + \cdots + X_n \geq (1+\epsilon)\mu] = \mathbf{P}\Big[e^{t(X_1+\cdots+X_n)} \geq e^{t(1+\epsilon)\mu}\Big] \overset{\text{(a)}}{\leq} \frac{\mathbf{E}\Big[e^{t(X_1+\cdots+X_n)}\Big]}{e^{t(1+\epsilon)\mu}}$$

by (a) Markov's inequality. Next analyze $\mathbf{E}\Big[e^{\epsilon X_1 + \cdots + X_n}\Big]$. We first have

$$\mathbf{E}\Big[e^{t(X_1+\cdots+X_n)}\Big] = \prod_{i=1}^{n} \mathbf{E}\Big[e^{tX_i}\Big]$$

by independendence of the X_i's. For each i, we have

$$e^{tX_i} \leq 1 + tX_i + t^2 X_i^2 \leq 1 + \Big(t + t^2\Big) X_i$$

where we recall that $0 \leq t, X_i \leq 1$ and apply the inequality $e^x \leq 1 + x + x^2$ for $x \leq 1$. Consequently

$$\mathbf{E}\Big[e^{tX_i}\Big] \leq \mathbf{E}\Big[1 + \Big(t + t^2\Big)X_i\Big] \leq 1 + \Big(t + t^2\Big)\,\mathbf{E}[X_i] \leq e^{(t+t^2)\,\mathbf{E}[X_i]},$$

where the last inequality follows from $1 + x \leq e^x$ for all x. Now we have

$$\mathbf{E}\Big[e^{t(X_1+\cdots+X_n)}\Big] \leq \prod_{i=1}^{n} \mathbf{E}\Big[e^{tX_i}\Big] \leq \prod_{i=1}^{n} \mathbf{E}\Big[e^{(t+t^2)\,\mathbf{E}[X_i]}\Big] \leq e^{(t+t^2)\mu},$$

hence

$$\mathbf{P}[X_1 + \cdots + X_n \geq (1+\epsilon)\mu] \leq e^{(t+t^2)\mu - t(1+\epsilon)\mu} = e^{t(t-\epsilon)\mu}.$$

The RHS is minimized by setting $t = \epsilon/2$, giving $e^{-\epsilon^2\mu/4}$.

The proof for the second inequality is similar and we provide a sketch highlighting a differences. Let $t \in (0,1]$ by a parameter TBD. We have

$$\begin{aligned}\mathbf{P}[X_1 + \cdots + X_n \leq (1-\epsilon)\mu] &= \mathbf{P}\Big[e^{-t(X_1+\cdots X_n)} \geq e^{-(1-\epsilon)t\mu}\Big] \\ &\leq \mathbf{E}\Big[e^{-t(X_1+\cdots+X_n)}\Big]e^{(1-\epsilon)t\mu}\end{aligned}$$

Note the introduction of the negative sign to reverse the inequality before applying Markov's inequality. As before, we apply independence to break the expectation into individual moments $\mathbf{E}\Big[e^{-tX_i}\Big]$, which are each bounded above by

$$\mathbf{E}\Big[e^{-tX_i}\Big] \leq 1 - \Big(t - t^2/2\Big)\,\mathbf{E}[X_i] \leq e^{-\left(t - t^2/2\right)\mathbf{E}[X_i]},$$

where we applied the inequality $e^x \leq 1 + x + x^2/2$ for $x \leq 0$. This leads to

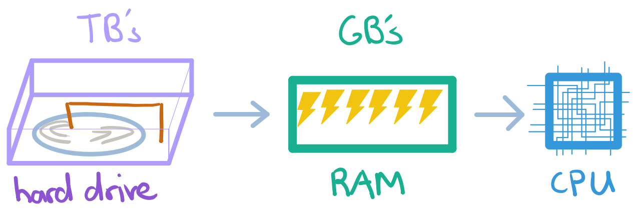
$$\mathbf{P}[X_1 + \cdots + X_n \leq (1-\epsilon)\mu] \leq e^{(t^2/2 - \epsilon t)\mu};$$

The RHS is minimized by $t = \epsilon$, giving the desired bound. □

Chapter 7

Online algorithms

7.1 Caching

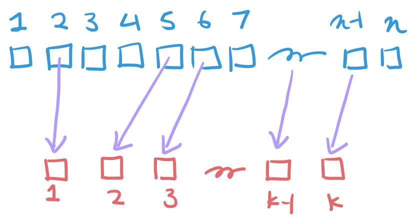


At a very high level, a computer consists of memory holding data and a CPU that computes the data. In one generic step, it loads a few bytes of data from memory into the CPU registers, does some calculations on the registers, and writes the result back into memory.

Memory is supplied by a series of devices with different tradeoffs between memory size and latency. The biggest device is the hard disk, which can hold terabytes of nonvolatile memory but is slow to read and write. Next we have RAM, which holds gigabytes of volatile memory, and is much faster than the the hard drive. Finally the CPU also has its own local (L_1 and L_2) cache, which holds at most a few megabytes of memory, but is extremely fast to access due to its proximity to the CPU.

At the end of the day, the memory is backed by the hard disk, and the RAM and local cache act as faster intermediate layers. When the CPU needs a piece of memory x, it first checks the local cache. If the local cache does not have x, then it checks the RAM. If x is not in RAM, then finally the computer retrieves x from disk. The RAM and the cache are both populated with x as it travels from hard disk to CPU. In the likely event that we read or write x again, it will already be cached in faster memory.

Hitting the hard disk throttles the computation. A good *cache strategy* tries to maximize the odds of the requested data already sitting in fast memory, so we can avoid hitting the disk. The main design decision is in choosing which data to *evict* from fast memory. In the example, when x was placed in memory, we needed to evict another item y to make room for x. Different strategies for choosing y can make a big difference in performance.



The model. We design and analyze cache strategies in the following simplified model. We assume there are n pieces of data of identical size, and k cache slots that can each hold one piece of data. In general, n is much larger than k. The input is an *online* sequence of data requests, in the form of indices out of $[n]$. Each request i is served from cache, so if i is not already in one of the k slots, then we have to choose which of the k slots to place it in. The general goal is to minimize the number of cache misses.

A salient point is that the requests are made *online*: we have to choose which item to evict without any knowledge of future requests.

The only design decision is in the *eviction policy*: when the requested data x is not in cache, which item y should be evicted to make space for x? Here are a few approaches:

- *Least frequently used (LFU):* Evict the item y that has been accessed the fewest number of times (since being put in cache).
- *Least recently used (LRU):* Evict the item y with the oldest time of last access.
- *Not recently used (NRU), a.k.a. 1-bit LRU:* Whenever an item is accessed, mark it. Evict unmarked items. If all cache items are marked, then remove all marks and try again.

LRU can be seen as a special case of NRU.

Competitive analysis. We typically analyze algorithms from a worst-case point of view. For caching, the worst-case perspective would try to bound the total number of cache misses. In the online model, however, the worst-case is unbounded: the adversary can always request an item out of cache and force a cache miss.

So maybe there is no good cache strategy because the data requests are simply impossible to cache. But *if there is a good cache strategy, then we can find a comparably good one?* Suppose that for a fixed sequence of data requests $i_1, i_2, \dots \in \mathbb{N}$, it was possible to have m cache misses *in hindsight*. Is it possible for us, operating online, to also get m misses? $100m$ misses? mk misses? etc. In competitive analysis, we want to minimize the competitive ratio,

$$\text{competitive ratio} \overset{\text{def}}{=} \frac{\text{our misses}}{\text{OPT misses}}$$

where OPT is the minimum number of misses in hindsight.

Least frequently used (LFU). Least frequently used has an unbounded competitive ratio. We leave the proof to the reader as an exercise.

Exercise 7.1. Prove that LFU has unbounded competitive ratio. (That is, for all $L > 0$, give a sequence of requests for which LFU obtains a competitive ratio $\geq L$.)

Not-recently-used (NRU) and least-recently-used (LRU). Next we analyze the not-recently-used and least-recently-used eviction strategies.

Theorem 7.1. *NRU and LRU have competitive ratio at most k.*

Proof. LRU is a special case of NRU, so we only need to prove the bound for NRU. Consider a sequence of requests

$$i_1, i_2, i_3, \dots \in [n].$$

We split the sequence whenever NRU resets of all of its marks. We call the contiguous subsequence of requests between resets a *run*.

Each run has k distinct items. The first item after a run is distinct from the k distinct items in the run. Thus: *any* eviction strategy must miss either (a) one of the last $k - 1$ distinct requests of the run, or (b) the first item after the run. That is, OPT makes at least 1 mistake per run.

On the other hand, NRU makes k mistakes per run. All put together, we have

$$\frac{\text{\# NRU misses}}{\text{\# OPT misses}} \leq \frac{k(\text{\# runs})}{\text{\# runs}} = k,$$

as desired. □

Randomized NRU. Lastly we consider a *randomized* variation of the NRU algorithm. It starts from the NRU framework: marking items as they are accessed, evicting unmarked items, and unmarking all items when all items are marked. The key distinction is that when evicting an unmarked item, it chooses one uniformly at random.

Theorem 7.2. *Randomized NRU has competitive ratio at most $2\ln(k)$ in expectation.*

Proof. As before, consider a sequence of requests

$$i_1, i_2, i_3, \dots \in [n],$$

and split the sequence into *runs* whenever all the marks are reset. There are k distinct items per run. Of these k items we distinguish two types:

1. "New" items that were not in the previous run.

2. "Repeat" items that were also in the previous run.

We define a potential function Φ equal to the number of items that are in OPT's cache, but not in the NRU cache.

Fix a single run. Let Φ_{IN} and Φ_{OUT} be the value of Φ at the beginning and the end of the run, respectively. Let ℓ be the number of new items in the run; that leaves there are $k - \ell$ repeat items in the run.

Consider OPT. At the beginning of the run, out of ℓ new pages not in the NRU cache, at most Φ_{IN} of them are in OPT's cache. So OPT misses at least $\ell - \Phi_{\text{IN}}$ new pages.

Now consider Φ_{OUT}. Φ_{OUT} is the number of items in OPT's cache but not in the NRU cache at the end of the run. But the NRU cache is filled with the k items in the run. So OPT must have kicked out these Φ_{OUT} items, and incurred Φ_{OUT} cache misses, during the run.

So OPT has at least $\ell - \Phi_{\text{IN}}$ misses, and at least Φ_{OUT} misses. Average these together, and OPT has at least

$$\frac{\ell + \Phi_{\text{OUT}} - \Phi_{\text{IN}}}{2}$$

misses in the run.

Now consider NRU. The NRU cache has ℓ cache misses for the ℓ new items. Consider the repeat items, which occupy $k - \ell$ of the k slots, all unmarked, at the beginning of the run. We claim that for $i = 1, \dots, k - \ell$, the ith repeat item is missed with probability at most $\frac{\ell}{k-(i-1)}$.

Consider the first repeat item. In the worst case, all ℓ new items come first, and they evict ℓ of the k items uniformly at random. The first item is evicted, hence missed, with probability ℓ/k.

Now consider the ith repeat item. When this request arrives, the cache contains:

(a) (At most) ℓ new items, all marked.

(b) The first $i - 1$ repeat items, all marked.

(c) $k - \ell - (i - 1)$ items out of the $k - (i - 1)$ from the previous run that have not appeared in this run, all unmarked.

Consider the unmarked items in the third category. Of the $k - (i - 1)$ items from the previous run that have not appeared in this run, exactly ℓ have been evicted. By symmetry, each has an equally likely chance of being evicted. This includes the ith repeat item. Thus the ith repeat item has a $\frac{\ell}{k-(i-1)}$ probability of being evicted, hence missed, when it is requested.

By linearity of expectation,

$$\begin{aligned}\mathbf{E}[\text{\# misses on repeats}] &= \sum_{i=1}^{k-\ell} \mathbf{P}[\text{missing on the } i\text{th repeat item}] \\ &\overset{\text{(a)}}{\leq} \sum_{i=1}^{k-\ell} \frac{\ell}{k+1-i} \leq \ell \ln(k).\end{aligned}$$

To review, in a single run, OPT makes at least $.5(\ell + \Phi_{\text{OUT}} - \Phi_{\text{IN}})$ mistakes, and randomized NRU makes at most $\ell \ln(k)$ mistakes in expectation.

Let L denote the total number of new items over the entire sequence. Over all the requests, the number of OPT misses is at least $L/2$ plus half of the total change in Φ. Initially $\Phi = 0$ and at the end it is nonnegative so the total change is nonnegative. Thus OPT makes at least $L/2$ mistakes.

Meanwhile the randomized NRU has $L \ln(k)$ cache misses in expectation. Thus the *expected* competitive ratio is bounded by

$$\mathbf{E}[\text{competitive ratio}] \leq \frac{L \ln(k)}{L/2} = 2 \ln(k),$$

as desired. □

7.2 Buy-or-rent

Someone without a car can use Lyft or Uber to get around. But after a lot of taxi fares, buying a car becomes more appealing. Should you buy, or continue to rent? It's tricky because you don't entirely know what the future holds.

Another example is with skis. The ski resort rents skis, but its not very cheap. You can also buy skies, which is more expensive than renting, but could save you money in the long run if you ski a lot. Should you rent or buy skis? (IMO its much harder to project future ski use then car use.)

Decisions like this pop up all over a computer. Should you keep a hard drive spinning when not in use? It takes a while to spin up a hard drive from rest. On the other hand it takes energy to keep it spinning and ready. The right decision depends on future data access patterns that you don't know. Something similar can be said for any component with a high startup cost.

For fun we will frame the problem in terms of skis. Suppose it costs k dollars to buy skis, and 1 dollar a day to rent skis. Let ℓ be the total number of days skied, in hindsight. The tricky part is that we don't know ℓ. Every day we decide to buy or rent skis, not knowing if we will ever ski again.

If we knew ℓ, the decision would be trivial:

1. If $\ell \leq k$, then rent every day.
2. If $\ell \geq k$, then buy on the first day.

Thus OPT $= \min\{k, \ell\}$ in hindsight. Our goal is to develop an *online strategy* competitive with OPT.

As a warmup, we challenge the reader to think of a 2-competitive algorithm.

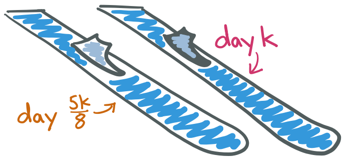
Exercise 7.2. Design and analyze an online, $(2 - o(1))$-competitive algorithm for the ski-rental problem on the preceding page.

In fact 2 is optimal for deterministic algorithms:

Exercise 7.3. Prove that $2 - o(1)$ is the optimal competitive ratio for any deterministic algorithm for the ski-rental problem on the previous page.

Half-skis. As a thought experiment, supposed we relaxed the rules so that you could buy one ski at a time for $k/2$ dollars, and rent 1 ski at a time for $1/2$ dollars. (You still need 2 skis each day). Note that the offline optimal strategy does not change: you should buy or rent (both) depending on if $\ell \geq k$ or not. Thus OPT $= \min\{k, \ell\}$. But here one can get a competitive ratio better than 2. Let us assume k is divisible by 8 for simplicity.

1. On day $5k/8$, buy one ski.
2. On day k, buy a second ski.



For $\ell < 5k/8$, we are optimal since we are only renting. For $\ell = 5k/8$, we pay

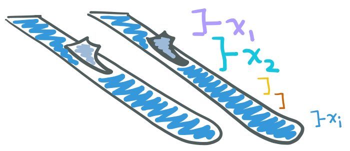
$$\frac{5k}{8} + \frac{k}{2} = \frac{9k}{8} = \frac{9}{5}\,\text{OPT}\,.$$

For $\ell = k$, we pay

$$\frac{5k}{8} + \frac{1}{2}\frac{3k}{8} + k = \frac{29}{16}k \leq \frac{29}{16}\,\text{OPT}\,.$$

For ℓ in between $5k/8$ and k, the competitive ratio is only better than at $5k/8$ or at k.

The takeaway is that operating "fractionally" – here we allow for "half" purchases – leads to a better ratio.



Liquid skis. Suppose we pushed the thought experiment further and allowed you to buy or rent arbitrary fractions of a pair of skis. For each day i, let x_i denote the fractional skis bought on day i. Let y_i be the fractional skis rented on day i. We require 1 total ski on each day i:

$$x_1 + \cdots + x_i + y_i \geq 1 \text{ for all } i \in [\ell].$$

Our goal is to minimize the total cost,

$$k(x_1 + \cdots + x_\ell) + y_1 + \cdots + y_\ell.$$

Observe that the optimum policy in hindsight is the same, hence $\text{OPT} = \min\{k, \ell\}$.

Here we will analyze the following heuristic. Let $\delta > 0$ be a parameter TBD. Suppose we commit to spending $1 + \delta$ units per day until we've completely bought the skis. Note that $y_i = 1 - x_1 - \cdots - x_i$ so x_i determines y_i. For day 1,

$$1 + \delta = kx_1 + (1 - x_1) \implies x_1 = \frac{\delta}{k-1}$$

For day 2,

$$1 + \delta = kx_2 + (1 - x_1 - x_2) \implies x_2 = \frac{1}{k-1}(\delta + x_1).$$

In general, on the ith day,

$$1 + \delta = kx_i + (1 - x_1 - \cdots - x_i) \implies x_i = \frac{1}{k-1}(\delta + x_1 + \cdots + x_{i-1}).$$

For the first k days, OPT pays 1 dollar per day, while we pay δ. So for $\ell \leq k$, we have a competitive ratio of $1 + \delta$. We now choose δ to ensure we own a full set of skis after k days, which guarantees that the competitive ratio is $1 + \delta$ for all ℓ. We have

$$\begin{aligned} 1 = x_1 + \cdots + x_k &= \frac{\delta}{k-1}\left(1 + \left(\frac{k}{k-1}\right) + \left(\frac{k}{k-1}\right)^2 + \cdots + \left(\frac{k}{k-1}\right)^{k-1}\right) \\ &= \delta\left(\left(\frac{k}{k-1}\right)^k - \left(\frac{k}{k-1}\right)\right). \end{aligned}$$

Rearranging, we have

$$\delta = \frac{1}{\left(\frac{k}{k-1}\right)^k - \left(\frac{k}{k-1}\right)},$$

and

$$\lim_{k\to\infty} \delta = \frac{1}{e-1} \approx .582.$$

Thus the competitive ratio $1+\delta$ converges to $\frac{e}{e-1} \approx 1.582$.

Random skis. Our improved competitive ratio is artificial since of course we cannot buy skis in arbitrary fractions. We convert the continuous strategy above to a fractional one by *randomized rounding.* The high-level idea is that we commit to buying in one of the first k days, and for $i \in [k]$, interpret x_i as the probability of buying skis on day i. This can be implemented by as follows.

1. Pick $\alpha \in [0,1]$ uniformly at random.
2. Simulate the fractional algorithm, giving values $x_1, x_2, \dots \in [0,1]$.
3. On day i, if $x_1 + \dots + x_i \geq \alpha$, buy skis.

The expected cost from buying skis is

$$\mathbf{E}[\text{cost buying}] = k \sum_{i=1}^{\ell} \mathbf{P}[\text{buy on day } i] = k(x_1 + \dots + x_\ell).$$

The probability we rent on day i is

$$\mathbf{P}[\text{rent on day } i] = \mathbf{P}[\alpha > x_1 + \dots + x_i] = 1 - x_1 - \dots - x_i = y_i.$$

Thus the expected cost from renting is

$$\mathbf{E}[\text{cost renting}] = \sum_{i=1}^{\ell} \mathbf{P}[\text{rent on day } i] = y_1 + \dots + y_\ell.$$

Thus the total expected cost is

$$\begin{aligned} \mathbf{E}[\text{cost}] &= \mathbf{E}[\text{cost buying}] + \mathbf{E}[\text{cost renting}] \\ &= k(x_1 + \dots + x_\ell) + y_1 + \dots + y_\ell, \end{aligned}$$

the same cost as our (impractical) fractional solution! Thus we have the same competitive ratio in expectation:

$$\mathbf{E}[\text{competitive ratio}] = 1 + \delta \xrightarrow{k\to\infty} \frac{e}{e-1}.$$

In conclusion:

Theorem 7.3. *There is randomized algorithm for the ski-rental problem with competitive ratio converging to $\frac{e}{e-1}$ for large k.*

7.3 Additional notes and materials

The analysis of deterministic NRU/LRU is from [ST85a]. The randomized NRU algorithm is from [FKL+91]. Additional notes on caching can be found in [Blu00] and [MR95, Chapter 13]. The buy-or-rent algorithm is from [KMM+94].

Lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

7.4 Exercises

Exercise 7.1. Prove that LFU has unbounded competitive ratio. (That is, for all $L > 0$, give a sequence of requests for which LFU obtains a competitive ratio $\geq L$.)

Exercise 7.2. Design and analyze an online, $(2 - o(1))$-competitive algorithm for the ski-rental problem on page 118.

Exercise 7.3. Prove that $2 - o(1)$ is the optimal competitive ratio for any deterministic algorithm for the ski-rental problem on page 118.

Exercise 7.4. Let $\ell < k$. Prove that an LRU cache of size k is $k/(k+1-\ell)$-competitive with any cache of size ℓ.

That means, for example, that an LRU cache of size k is 2-competive with any cache of size $k/2$. Some people find this bound more compelling.

You should be able to prove this by a short modification of the argument of the k-competitive bound. It suffices to point out which part of the argument should change, and how.

Exercise 7.5. Suppose you are going to the graduate student social thingy on the 3rd floor of Lawson.[1] You can get to the third floor by either an elevator or the stairs. The elevator takes 15 seconds (once you get in), while the stairs take 2 minutes. Your goal is to get up to the third floor as fast as possible before the donuts are all taken.

You press the button to go up for the elevator. You don't know how long it will take to come down. Do you wait or take the stairs?

1. Suppose you knew how long the elevator would take to arrive. What is the optimal choice, based on this wait?
2. Now suppose you don't know how long the elevator would take. Design a deterministic algorithm that is (15/8)-competitive with the optimal solution (where you know how long the elevator would take).
3. Suppose you have a quarter in your pocket, which lands heads or tails with equal probability. You can toss the coin once every 15 seconds. Design a randomized algorithm with a (slightly) better competitive ratio than the (best) deterministic one from the previous question.[2]

[1](It used to be on the third floor.)

[2]I actually don't know what the best competitive ratio would be, and I'm interested to see what everyone comes up with.

Chapter 8

Distinct elements and moment estimation

Let $e_1, \ldots, e_m \in [n]$ be a sequence of elements from an universe of n possible elements. For each element $i \in [n]$, let $f_i = |\{j : e_j = i\}|$ denote the frequency of element i. For $k \in \mathbb{Z}_{\geq 0}$, the kth moment is the sum F_k defined by

$$F_k = \sum_{i=1}^{n} f_i^k.$$

For example, the zeroth moment,

$$F_0 = \sum_{i=1}^{n} f_i^0 = |\{i : f_i > 0\}|$$

counts the number of distinct elements in the sequence. The number of distinct elements of course has numerous applications; for example, Google analytics tracks the number of distinct visitors to each website [HNH13]. In databases, quick estimates of the number of distinct elements are used to optimize complex queries [HNS+95].

The second moment,

$$F_2 = \sum_{i=1}^{n} f_i^2$$

is called the "repeat rate" or "Gini's index of homogeneity" and is used to compute the surpise index [Goo89]. In general, frequency moments F_k for $k \geq 2$ reflect the *skew* of the data, which (for example) is used by databases to select data partitioning schemes (cf. [Yua92]).

Clearly moment estimation is easy to do in $O(n)$ space, since we can count each f_i explicitly. Here, in the same spirit as in the heavy hitters problem (Chapter 2), we are interested sublinear-space algorithms, processing the elements $e_1, \ldots, e_m$ online. As before, we assume the stream is generated adversarially.

We discuss algorithms for F_0 and F_2 in Section 8.1 and Section 8.2, respectively. An estimator for F_k is described in exercise C.17.

For simplicity, we assume that n and m are within a polynomial factor of each other, so that $1/\operatorname{poly}(n)$ represents a high-probability bound over both the total number of elements, and the length of the stream.

8.1 F_0: Distinct elements

Consider the problem of estimating the number of distinct elements, F_0, in the stream. Of course this is easy in $O(n)$ space by explicitly maintaining the set of all distinct elements. But in sublinear space we cannot store the identify of all the distinct elements when we've seen. When the next element comes in, how do we know if it is new? Any ideas?

Suppose we had a subroutine that, given a parameter $p \in [0,1]$, maintained a random sample B including each distinct element independently with probability p. Then $\mathbf{E}[|B|] = pF_0$, making $|B|/p$ an unbiased estimate for λ. $|B|$ is also a sum of independent random $\{0,1\}$-variables indicating whether each distinct element is sampled. Thus, if $\mathbf{E}[|B|] = pF_0$ is at least $\Omega(\log(n)/\epsilon^2)$, then $|B|/p$ is a $(1 \pm \epsilon)$-approximation of F_0 with high probability.

(Indeed, for each element i, let $X_i = 1$ if i is sampled, and 0 otherwise. We have $\mathbf{E}[X_i] = 1$ for each distinct element i, hence $|B| = \mathbf{E}[\sum_{i=1}^{n} X_i] = pF_0$. The Chernoff bound states that $\mathbf{P}[||B| - pF_0| \geq \epsilon pF_0] \leq e^{-\Omega\left(\epsilon^2 pF_0\right)}$.)

For example, if p satisfies

$$\frac{C\log(n)}{\epsilon^2} \leq pF_0 \leq \frac{2C\log(n)}{\epsilon^2}$$

for a sufficiently large constant C, then $(1-\epsilon)F_0 \leq |B|/p \leq (1+\epsilon)F_0$ with high probability. Moreover, $|B| \leq (1+O(\epsilon))pF_0 \leq O(\log(n)/\epsilon^2)$, satisfying our low-space requirements.

To actually implement this strategy we have two challenges to address:

1. Given a probability p, how can we maintain a sample B where each distinct element is sampled with probability p, online and in low-space?

2. How can we choose the right value p – big enough that $|B|/p$ is a good estimate with high probability, but small enough that $|B|$ isn't too large?

Consider the first task. Imagine stepping through the stream one element at a time. The first element e_1 certainly should be sampled and added to B with probability p.

What about the second element e_2? If e_2 is different from e_1, then e_2 should also be added to B with probability p. What if e_2 is the same as e_1? We should skip it, since we already tried sampling that element with probability p.

But it's not so simple. In later iterations i, we wouldn't know if we already tried to sample e_i in a previous iteration, unless it was actually sampled and placed in B.

Any other ideas?

Here's the twist: on the ith element e_i, we can *invalidate* any previous attempt to sample e_i by removing it from B if it is already in B. Then B samples every distinct element except e_i with probability p. Now sample e_i and add it to B with probability p. In effect, B is sampling the *last* instance of each distinct element independently with probability p. See Fig. 8.1 for pseudocode.

We now know how to maintain a random sample B of each distinct element with probability p. The only problem now is that we need the right value p (up to a constant factor), inversely proportional to F_0, so that $|B|$ is large enough to provide a reliable estimate, but small enough to have a good space bound. Any ideas?

Let $\tau = C \log(n)/\epsilon^2$ for a sufficiently large constant C. Initially, we start with $p = 1$, and maintain the set of distinct elements explicitly in a buffer B until $|B| \geq \tau$. Then we set $p = 1/2$, and drop every element in B independently with probability $1/2$. Continuing in this fashion, every time we reach $|B| \geq \tau$, we divide p in half and drop each element in B. By randomly dropping elements from B in lockstep with decreases to p, we maintain the invariant that B samples every distinct element with probability p. In particular, $|B|/p$ always estimates the number of distinct elements. See Fig. 8.2 for pseudocode.

```
distinct-sample($p$)
1. Let $B = \emptyset$
/* $|B|/p$ is our estimate for the number of distinct elements. */
2. For each element $e_i$ (for $i = 1, 2, ...$):
    A. If $e_i \in B$ then remove $e_i$ from $B$.
    B. With probability $p$, add $e_i$ to $B$.
```

Figure 8.1: Maintaining an independent sample of each distinct element with probability p.

To analyze this algorithm, one can reimagine it as the following equivalent process. For each $j \in \{0, \dots, \lceil \log n \rceil\}$, let B_j maintain a random sample where each distinct element is sampled with probability $1/2^j$. We correlate the samples B_j across j by imagining, for each element e_i in the stream, drawing a random number $\theta_i \in [0, 1]$, and adding e_i to B_j if $\theta_i \leq 1/2^j$.

Consider the probability p and buffer B in our actual algorithm at any fixed point in the stream. In terms of the virtual buffers B_j, we have $B = B_j$ and $p = p_j$ for the first index j where $|B_j| < \tau$ all throughout the stream.

We claim that with high probability, for all indices j and all points in the stream:

(a) If $p_j F_0 \geq \tau/4$, then $(1 - \epsilon)F_0 \leq |B_j|/p_j \leq (1 + \epsilon)F_0$.

(b) If $|B_j| \geq \tau$, then $p_i F_0 \geq \tau/2$.

Suppose these properties held and fix any point in the stream. The second property implies that $p = p_j$ for an index j such that $p_j F_0 \geq \tau/4$ (since $|B_{j-1}| \geq \tau$ at some point in the stream). The first property then implies that $F_0/p = F_0/p_j$ is an $(1 \pm \epsilon)$-estimate of F_0, as desired.

It remains to prove properties (a) and (b). Property (a) is a direct application of the Chernoff bound. Property (b) is the contrapositive of Chernoff bound. Indeed, suppose $p_j F_0 \leq \tau/2$. Let k be the largest index with $p_k F_0 \leq \tau/2$. We have $\tau/4 \leq \mathbf{E}[B_k] = p_k F_0 \leq \tau/2$. Since $B_j \subseteq B_k$, we have

$$\mathbf{P}[|B_j| \geq \tau] \leq \mathbf{P}[|B_k| \geq \tau] \leq \mathbf{P}[|B_k| \geq 2p_k F_0] \leq e^{-\Omega(p_k F_0)} = 1/\operatorname{poly}(n).$$

By the union bound, with high probability, $|B_j| \leq \tau$ for all j whenever $p_j F_0 \leq \tau/2$. This implies property (b) and completes our analysis.

Theorem 8.1. *The algorithm* `distinct-elements` *maintains a $(1 \pm \epsilon)$-approximation to the number of distinct elements with high probability in $O(\log(n)/\epsilon^2)$, assuming each element can be stored in $O(1)$ space.*

```
distinct-elements($n$,$\epsilon$)
1. Set $p = 1$, $B = \emptyset$, and $\tau = C\log(n)/\epsilon^2$ for a sufficiently large constant $C$.
/* $|B|/p$ is our estimate for the number of distinct elements. */
2. For each element $e_i$ (for $i = 1, 2, ...$):
   A. If $e_i \in B$ then remove $e_i$ from $B$.
   B. With probability $p$, add $e_i$ to $B$.
   C. If $|B| \geq \tau$:
      1. $p \leftarrow p/2$.
      2. Drop each element in $B$ independently with probability $1/2$.
```

Figure 8.2: A randomized streaming algorithm estimating the number of distinct elements.

8.2 F_2 estimation

We now shift to esimating the second moment, $F_2 = \sum_{i=1}^{n} f_i^2$. Any ideas?

Let $h : [n] \to \{-1, 1\}$ be a k-wise independent hash function assigning a random sign to each elements for a parameter k TBD. (Let's imagine h is an ideal hash function for now, and reverse engineer the right value of k afterwards.) Consider the estimate

$$Y^2 \text{ where } Y \overset{\text{def}}{=} \sum_{i=1}^{n} h(i) f_i.$$

Its easy to maintain Y in an online and small space fashion, simply adding $h(e_j)$ to Y for each item e_j.

We start with the expected value of Y^2. We have

$$\begin{aligned}\mathbf{E}\Big[Y^2\Big] &\overset{\text{(a)}}{=} \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{E}[h(i)h(j)] f_i f_j \\ &= \sum_{i=1}^{n} \mathbf{E}\Big[h(i)^2\Big] f_i^2 + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[h(i)h(j)] f_i f_j\end{aligned}$$

where (a) is by linearity of expectation. For the first sum, we have

$$\sum_{i=1}^{n} \mathbf{E}\Big[h(i)^2\Big] f_i^2 = F_2$$

because $h(i)^2 = 1$ (always). For the second sum, we have

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{E}[h(i)h(j)] f_i f_j = 0$$

because

$$\mathbf{E}[h(i)h(j)] = \mathbf{E}[h(i)]\,\mathbf{E}[h(j)] = 0$$

by pairwise independence. Thus $\mathbf{E}[Y^2] = F_2$, and Y^2 is an unbiased estimate of F_2.

The fact that Y^2 is unbiased is a nice first step, but how *reliable* is the estimate? How much does it deviate from its mean?

For a random variable X, the *variance* of X, denoted $\mathrm{Var}[X]$, is the expected squared difference between X and its mean:

$$\mathrm{Var}[X] \overset{\text{def}}{=} \mathbf{E}\Big[(X - \mathbf{E}[X])^2\Big].$$

It is not difficult to show that

$$\mathrm{Var}[X] = \mathbf{E}\Big[X^2\Big] - \mathbf{E}[X]^2$$

(see exercise C.13).

Consider the variance of our unbiased estimate Y^2. We have

$$\mathrm{Var}\Big[Y^2\Big] = \mathbf{E}\Big[Y^4\Big] - \mathbf{E}\Big[Y^2\Big]^2 = \mathbf{E}\Big[Y^4\Big] - F_2^2.$$

By linearity of expectation,

$$\mathbf{E}\Big[Y^4\Big] = \mathbf{E}\left[\left(\sum_{i=1}^{n} h(i) f_i\right)^4\right] = \sum_{i,j,k,\ell=1}^{n} \mathbf{E}[h(i)h(j)h(k)h(\ell)] f_i f_j f_k f_\ell.$$

Assume for simplicity that h is an ideal hash function, and consider an individual summand $\mathbf{E}[h(i)h(j)h(k)h(\ell)] f_i f_j f_k f_\ell$. The indices i, j, k, ℓ may or may not be distinct.

Suppose first that i, j, k, ℓ are all distinct. Then

$$\mathbf{E}[h(i)h(j)h(k)h(\ell)] = \mathbf{E}[h(i)]\,\mathbf{E}[h(j)]\,\mathbf{E}[h(k)]\,\mathbf{E}[h(\ell)] = 0.$$

More generally, if any index appears as an odd power, then the product of hashed values is again negative. For example, if i was distinct and j, k, ℓ have the same value, like $i = 1$ and $j = k = \ell = 2$, then

$$\mathbf{E}\Big[h(i)h^3(j)\Big] = \mathbf{E}[h(i)]\,\mathbf{E}\Big[h^3(j)\Big] = 0.$$

Similarly, if we have three distinct indices, like $i = 1$, $j = 2$, and $k = \ell = 3$, then

$$\mathbf{E}\Big[h(i)h(j)h^2(k)\Big] = \mathbf{E}[h(i)]\,\mathbf{E}[h(j)]\,\mathbf{E}\Big[h^2(k)\Big] = 0.$$

The only nonzero terms are those where each index appears as an even power. Here we have two types. The first is when all the indices are identical. We have

$$\mathbf{E}\Big[h^4(i)\Big] f_i^4 = f_i^4$$

since $h^4(i) = 1$. The second case is where we have two distinct indices, squared. We have

$$\mathbf{E}\Big[h^2(i)h^2(j)\Big] f_i^2 f_j^2 = f_i f_j^2,$$

and each pair of indices appears $\binom{4}{2} = 6$ times.

Coming back to $\mathbf{E}[Y^4]$, we now have

$$\mathbf{E}\Big[Y^4\Big] = \sum_{i=1}^n f_i^4 + 6\sum_{i<j} f_i^2 f_j^2 \leq \left(\sqrt{3}\sum_{i=1}^n f_i^2\right)^2 = 3F_2^2.$$

Altogether, we have

$$\mathrm{Var}\Big[Y^2\Big] = 3F_2^2 - F_2^2 = 2F_2^2.$$

So the variance of our unbiased estimator Y_2^2 is twice the square of its expected value, F_2^2. This is assuming that h is an ideal hash-function. Looking back, it is clear that 4-wise independent hash functions suffice.

How does the variance relate to the absolute deviation, $|Y_2^2 - F_2|$? For $\alpha > 1$, we have

$$\mathbf{P}\Big[\Big|Y_2^2 - F_2\Big| \geq \alpha F_2\Big]\,\mathbf{P}\Big[(Y^2 - F_2)^2 \geq \alpha^2 F_2^2\Big] \overset{\text{(b)}}{\leq} \frac{\mathrm{Var}[Y^2]}{\alpha^2 F_2^2} \leq \frac{2}{\alpha^2}. \tag{8.1}$$

where (b) applies Markov's inequality to the squared deviation $(Y^2 - F_2)^2$.

For $\alpha = 2$, this inequality says that we have a factor 2 approximation with probability 1/2. But what if we wanted a more accurate estimate, like 10% error? The inequality gives us nothing for values of $\alpha \leq \sqrt{2}$. But consider the penultimate inequality:

$$\mathbf{P}\Big[\Big|Y^2 - F_2\Big| \geq \alpha F_2\Big] \leq \frac{\mathrm{Var}[Y^2]}{\alpha^2 F_2^2}.$$

The more general inequality,

$$\mathbf{P}[|X - \mathbf{E}[X]| \geq \lambda] \leq \frac{\operatorname{Var}[X]}{\lambda^2},$$

is called *Chebyshev's inequality*, follows from the same derivation as (8.1), and bounds the consistency of a random variable by its variance. For example, if we wanted a $(1 \pm \epsilon)$-approximation with constant probability, then Chebyshev's inequality says we need to take the variance down from $2F_2^2$ to $\epsilon^2 F_2^2$. How can we reduce the variance of our estimate Y^2?

Take the *average*. In general, given a random variable X, suppose we made k independent copies $X_1, \dots, X_k$ of X. Consider their average. It is not difficult to show that

$$\operatorname{Var}\left[\frac{1}{k}\sum_{i=1}^{k} X_i\right] = \frac{1}{k}\operatorname{Var}[X].$$

(See exercise C.39.) Thus, if we made k independent copies $Y_1^2, \dots, Y_k^2$ of our estimate Y^2, and took their average Z, we have

$$\operatorname{Var}[Z] = \frac{1}{k}\operatorname{Var}\left[Y^2\right] = \frac{2}{k}F_2^2.$$

In particular, for $k = 4/\epsilon^2$, we have $\operatorname{Var}[Z] \leq \epsilon^2 F_2^2/2$, and Chebyshev's inequality gives us

$$\mathbf{P}[|Z - F_2| \geq \epsilon F_2] \leq \frac{\operatorname{Var}[Z]}{\epsilon^2 F_2^2} \leq 1/2. \tag{8.2}$$

We now have an unbiased estimator that obtains a $(1 \pm \epsilon)$-approximation with constant probability, by taking the average of $O(1/\epsilon)$ copies of our initial estimator Y^2. What if we wanted to reduce the probability of error to a prescribed value δ? We could take the $O(1/\delta\epsilon^2)$ random variables instead, and the calculations in (8.2) would give probability of error δ instead of $1/2$. But what about the high-probability regime of $\delta = 1/\operatorname{poly}(n)$? Making $\operatorname{poly}(n)$ copies kills the entire space-spacing enterprise. The real question is: can we amplify *better than averaging*?

Let Z be the average of $4/\epsilon$ independent copies of Y^2. Then $|Z - F_2| \leq \epsilon F_2$ with probability (at least) $3/4$. Let $Z_1, \dots, Z_\ell$ be ℓ independent copies of Z. We expect at least $3/4$ of the Z_i's to be $(1 \pm \epsilon)$-approximations, and at most $1/4$ of the Z_i's to not. Now, the Chernoff bound implies that at most $1/2$ of the Z_i's are correct with probability exponentially small in ℓ.

(Indeed, if we define random variables $X_1, \dots, X_\ell$, where $X_i = 1$ if $|Y_i - \epsilon F_2| \geq \epsilon F_2$ and $X_i = 0$ otherwise, then $X_1, \dots, X_\ell$ are independent $\{0,1\}$ random variables with $\sum_{i=1}^{\ell} X_i \leq \ell/4$. Chernoff's bound now gives $\mathbf{E}\left[\sum_{i=1}^{\ell} X_i \geq \ell/2\right] \leq e^{-\ell/2}$.)

Set $\ell = O(\log(1/\delta))$. With probability $1-\delta$, at least half of the estimators Z_i are $(1 \pm \epsilon)$-approximations. Now, how can we extract an $(1 \pm \epsilon)$-approximation out of $Z_1, \dots, Z_\ell$, assuming half of them are $(1 \pm \epsilon)$-approximations?

Take the *median*. If half the Z_i's are correct, then their median is correct. (Less than half are too big, so the median isn't too big. Less than half are too small, so the median isn't too small.) Taking the median of $O(\log(1/\delta))$ estimates, each of which are the average of $O(1/\epsilon^2)$ counters, we obtain the following theorem.

Theorem 8.2. *One can compute an $(1 \pm \epsilon)$-approximation of the second moment F_2 with probability $1-\delta$ in $O(\log(1/\delta)/\epsilon^2)$ space assuming $O(1)$ space for each counter.*

The median-of-means trick. The F_2 estimator illustrates a more general amplification technique. In the general setting, we have an unbiased estimator X of some value μ, and the goal is to obtain and $(1+\epsilon)$-approximation with probability $1-\delta$. We amplify over two steps:

1. Let Y be the average of enough independent copies of X so that Y is a $(1 \pm \epsilon)$-approximation with probability strictly greater than $1/2$. (E.g., with probability $3/4$.)
2. Take the median of $O(\log(1/\delta))$ independent copies of of Y. With probability $1-\delta$, at least half the copies of Y are $(1 \pm \epsilon)$-approximations, hence so is the median.

This *"median-of-means"* can be much more efficient then just taking the average of many independent copies of X.

Note that the median of means may be a biased estimator, even though it is very accurate with high probability.

8.3 Additional notes and materials

The F_0 estimate in Section 8.1 is relatively recent and was given in [CVM22]. See also [Knu23]. The F_2 and F_k estimates are from [AMS99]. [AMS99] also included a different estimator for F_0. [AMS99] first appeared in 1996, and the techniques (sketching, median trick) and themes (randomization, low-space) had a strong influence on the subsequent literature of "big-data" streaming algorithms.

Lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

8.4 Exercises

Exercise 8.1. Prove that $\text{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$.

Exercise 8.2. Prove that given k independent copies $X_1, \dots, X_k$ of the same random variable X,

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

Exercise 8.3. Consider the special case of the distinct elements streaming problem where there are $n+1$ total items in a stream, each of which is one of n different possible items $\{1, \dots, n\}$. Show that any algorithm that maintains the number of distinct elements exactly throughout the stream has to use at least n bits of memory.

Exercise 8.4. A funny characteristic of the distinct elements estimator in Section 8.1 is that it is not monotone – $|B|$ can go down when the next element e_i kicks out a previously sampled copy of the same element – even though the true number of distinct elements is nondecreasing.

Design and analyze a streaming algorithm with the same performance as the distinct elements algorithm, with the additional property that the estimate is nondecreasing.

Exercise 8.5. We consider the problems of estimating the mean and the median of a stream of numbers. The input consists of a stream of integers $x_1, x_2, \dots,$ presented one at a time, of unknown length m. Your algorithms should use sublinear space and return $(1 \pm \epsilon)$-approximations (as defined below) of the desired statistics with probability at least $1 - \delta$.

We expect you to analyze the space, the running time, and the probability of outputting an accurate solution. The smaller the space, and in general the lower the dependency on ϵ and δ, the better. For simplicity you can assume it takes $O(1)$

space to store a number and it takes constant time for basic operations like adding, subtracting, and multiplying numbers.[1]

It may be helpful to be aware of a streaming algorithm called *reservoir sampling* that maintains a sample of 1 element drawn uniformly at random from the stream. This algorithm is easy to describe. It takes the first element deterministically. For $i > 1$, with probability $1/i$, it takes the ith element and replaces the element previously held by the algorithm. It is known (and you can assume as fact) that at any point in time, the element held by the algorithm represents an element sampled uniformly at random from the stream. Note that for any $k \in \mathbb{N}$, one can maintain a random sample of k elements with repetition from the stream by running k copies of the single-element reservoir sampling algorithm in parallel.

The two statistics we are interested in are defined as follows. Let $\delta, \epsilon \in (0,1)$ be fixed.

1. (2 points) A $(1 \pm \epsilon)$-approximation of the mean of the stream with probability at least $1 - \delta$. That is, if the mean is x, then you should return a value y in the interval $(1-\epsilon)x \leq y \leq (1+\epsilon)x$.

2. (8 points) A rank-wise $(1 \pm \epsilon)$-approximation of the median number in the stream with probability at least $1 - \delta$. That is, if the stream has m total numbers, then you should return an element whose rank r is in the interval $\lfloor (1-\epsilon)m/2 \rfloor \leq r \leq \lceil (1+\epsilon)m/2 \rceil$.[2]

Exercise 8.6. Recall that in the heavy hitters problem, the goal was to estimate the absolute frequency of each element (in $[n]$) up to an additive error of ϵm, where m is the total length of the stream. Another way to frame this to first let $x \in \mathbb{R}^n$ denote the frequency vector; that is, x_i is the absolute frequency of element i, and $\|x\|_1 = m$. We can think of `count-min` as estimating each coordinate x_i with (one-sided) additive error of $\epsilon\|x\|_1$.

In this problem we do something similar except with respect to the ℓ_2-norm. The goal is to estimate each coordinate x_i up to an additive error of $\pm\epsilon\|x\|_2$, and holds for real-valued $x \in \mathbb{R}^n$ (unlike `count-min`, which only holds for nonnegative vectors). Formally, we start with the all-zero vector $x = \mathbb{0}^n$. The stream presents data of the form (i, Δ), where $i \in [n]$ and $\Delta \in \mathbb{R}$, which indicates the update $x_i \leftarrow x_i + \Delta$. We want a data structure that can estimate each coordinate x_i up to $\pm\epsilon\|x\|_2$ with high probability, in sublinear space.

[1]Of course you are not allowed to abuse this with something bizarre like using a polynomial-bit integer as a bit map. This isn't battlecode.

[2]An element has rank i if it is the ith smallest element.

Below we describe a data structure that can get $\pm\epsilon\|x\|_2$ error for an appropriate choice of parameters. (This is like describing one "row" of the `count-min` data structure.) You will first be asked to choose the parameters and prove the error guarantee. Then you will be asked to amplify the data structure to obtain a high probability guarantee.

The data structure is as follows. Let $\epsilon > 0$ be given, let $w \in \mathbb{N}$ be a parameter TBD, and let $A[1..w]$ be an array of values initially set to 0. Let $h : [n] \to [w]$ be a pairwise independent hash function. Let $g : [n] \to \{-1, 1\}$ be a second pairwise independent hash function. The operations are as follows.

- For each update (i, Δ) presented by the stream, we set $A[h(i)] \leftarrow A[h(i)] + g(i)\Delta$.
- To retrieve an estimate for coordinate i, we return $g(i)A[h(i)]$.

We now analyze this approach, as follows.

1. For each i, let $y_i = g(i)A[h(i)]$ denote the estimate returned by the data structure. Prove that y_i is an unbiased of x_i for each i. (That is, $\mathbf{E}[y_i] = x_i$ for all i.)
2. What is the variance of y_i (as a function of w)?
3. Prove that for an appropriate choice of w, the probability that $|x_i - y_i| \geq \epsilon\|x\|_2$ is at most $1/10$. (w should depend on ϵ, and in general, the smaller the choice of w, the better. The choice of $1/10$ is arbitrary; any probability less than $1/2$ would suffice.)
4. Using the data structure designed above, design and analyze a data structure that, in $O(\log(n)/\epsilon^2)$ space, estimates each coordinate of x up to an additive error of $\pm\epsilon\|x\|_2$ with high probability. (I.e., probability of error at most $1/\operatorname{poly}(n)$.)

Exercise 8.7. This exercise is about estimating F_k for $k > 2$ with sublinear space as the elements are presenting in a stream (as described in the introduction). Let $g(x) = x^k$ for fixed x. Then $F_k = \sum_i g(f_i)$, where f_i is the frequency of item i. We break down the design and analysis of such an estimator into steps below, but you are encouraged to try to design one yourself first.

For each element $e \in [n]$, and each index in the stream $i \in [m]$, let $f_e^{(i)}$ be the frequency of element i after ith iterations. Consider the random variable Y defined by

$$Y \overset{\text{def}}{=} m\Big(g(f_{e_i}^{(m)} - f_{e_i}^{(i)}) - g(f_{e_i}^{(m)} - f_{e_i}^{(i)} - 1)\Big)$$

where $i \in [m]$ is drawn uniformly at random.

1. How can you implement Y in a streaming fashion? (In a stream, you don't know m).[3]

2. Prove that Y is an unbiased estimator for F_k.

3. Prove that the variance of Y is at most $kn^{1-1/k}F_k^2$.[4]

4. Using Y, design and analyze a streaming algorithm that computes a $(1+\epsilon)$-approximation for F_k with probability $1-\delta$, for given parameters ϵ and δ. In addition to the correctness, analyze the time and space of your algorithm.

Exercise 8.8. You run a double-secret laboratory and for your experiments you need to monitor the temperature of the lab very carefully. To this end you can buy thermostats $T_1, \ldots, T_k$ that purport to measure the temperature μ, but the thermostats are imperfect. You have the following facts.

1. Given the actual temperature μ of the lab, the thermostat readings T_i are independent.

2. Each thermostat is calibrated so that its expected value $\mathbf{E}[T_i]$ equals the actual temperature μ of the lab.

3. For each thermostat T_i, the variance $\mathrm{Var}[T_i]$ of the thermostat is σ^2 for a known parameter $\sigma > 0$.

Given parameters $\epsilon, \delta \in (0,1)$, the goal is to be able to measure the temperature of the room with additive error at most ϵ with probability at least $1-\delta$. Describe and analyze a system that, using as few thermostats as possible[5], obtains additive error ϵ with probability at least $1-\delta$.

[3] *Hint:* Exercise 2.5.

[4] *Hint:* This is a bit messy. One approach is to first show that $\mathrm{Var}[Y] \leq kF_1F_{2k-1}$, and then bound $F_1F_{2k-1} \leq n^{1-1/k}F_k^2$.

[5] up to constant factors independent of ϵ, δ, and σ

Chapter 9

Dimensionality Reduction

9.1 Large data sets and long vectors

Big data, big data, big data. What's so big about it? There are at least two dimensions to be aware of. First, there is a huge number of "pieces" of data being collected. For example, in the heavy hitters problem, we might have a piece of data for every search query ever made. A second dimension that we have not yet confronted is the "size" or "width" of each piece of data. Here we will consider data where each piece of data is a high-dimensional array of real values; i.e., points in $\mathbb{R}^d$ for d very large.

High-dimensional vectors arise naturally. Every graph is associated with a square *adjacency matrix* whose dimensions are proportional to the number of vertices, n. Thus every row is an n-dimensional vector. The world wide web and social networks are by now extremely large graphs where the corresponding vectors have very high dimension. In text processing, text is sometimes represented as a "bag-of-words", where one counts the frequency of each word. This can be encoded as a feature vector whose dimensionality is proportional to the size of the English language! (Plus typos.) To take this further, more aggressive algorithms use phrases — sequences of (say) 3 consecutive words — rather than words, and run algorithms on "bag-of-phrases" vectors. These vectors have dimension proportional to the size of the English language, *cubed*! A recent technique from machine learning, called *autoencoders*, first trains a large model (such as a neural network) on some large collection of data. For each piece of data, the internal state of the model when labeling that data is ultimately a high dimensional vector. It has been observed that these high-dimensional vectors can have useful geometries; e.g., in the `word2vec` tool for word embeddings [MSC+13].

We note that in some of the examples above, the data vectors are typically sparse with few nonzero entries. Such vectors can be represented more compactly as an "adjacency list", where we list the index and the value of only the nonzero entries in the vector. The trouble arises when we start running computations over them. When we start combining these vectors in some linear algebraic procedure, the vectors rapidly become dense, and this is where we pay for the high dimensions.

Most operations with vectors take time proportional to the number of dimensions (in the worst and dense case). Certainly it would be desirable for the data to live in a much lower dimensional space. The goal in this discussion is to develop some techniques for transforming high-dimensional data into lower-dimensional data. We first note that for many applications, we do not necessarily require the exact coordinates of the vector. Given a set P of points in a high-dimensional space d, we may only actually need the following:

1. For a given point $x \in P$, the (Euclidean) *length* of x, $\|x\| = \sqrt{\sum_i x_i^2}$.
2. For any two points $x, y \in P$, the Euclidean *distance* $\|x - y\|$ between them.
3. For any two points $x, y \in P$, the dot product $\langle x, y \rangle$ between x and y.

Moreover, for many applications, approximations to the above quantities may suffice to produce approximation algorithms for the given context.

We now introduce the main result of this chapter.

Theorem 9.1 (Johnson and Lindenstrauss [JL84])**.** *Let $P \subseteq \mathbb{R}^d$ be a collection of n points in $\mathbb{R}^d$, and let $k = O(\log(n)/\epsilon^2)$. Let $A \in \mathcal{N}^{k \times d}$ be a $k \times d$ randomized matrix where each coordinate is sampled as an independent Gaussian. Consider the randomly constructed linear map $f : \mathbb{R}^d \to \mathbb{R}^k$ defined by*

$$f(x) = \frac{1}{\sqrt{k}} Ax.$$

Then with probability of error $\leq 1/\operatorname{poly}(n)$, we have

$$(1 - \epsilon)\|x\| \leq \|f(x)\| \leq (1 + \epsilon)\|x\|$$

for all $x \in P$, and

$$(1 - \epsilon)\|x - y\| \leq \|f(x) - f(y)\| \leq (1 + \epsilon)\|x - y\|$$

for all pairs $x, y \in \mathbb{R}^d$.

This is a remarkable theorem. Theorem 9.1 says that, for the sake of preserving distances, one can always reduce the dimension to about $\log(n)$, where n is the number of input points. This bound is entirely independent of the input dimension. The input dimension could be as large as one could possibly imagine; the output dimension will always be a logarithmic function of the number of points. The construction, moreover, is *oblivious to the input.*

Perhaps even more remarkable is how *obvious* this mapping is after some acquaintance with Gaussian variables and their extremely convenient properties. The ideas underlying Theorem 9.1 lead to many other practical and simple (at least, to implement) algorithms, as we will see.

We note that the above guarantees also lead to approximations on pairwise dot-products; see exercise C.12.

We remark that the embedding A given in Theorem 9.1 is not particularly compact, since it requires an independent Gaussian. This could be formiddably expension. Here one can instead replace the Gaussian entries with $\{-1, 0, 1\}$ entries generated by appropriate hash functions [Ach01; DKS10; KN14; CJN18]. The intuition is similar because $\{-1, 0, 1\}$-random variables behave similarly to Gaussian's in a certain technical sense. (They are both *sub-Gaussian*; see [Ver18]). There is particular interest in ensuring that A is *column-sparse*, since this determines the running time of applying A. An alternative approach uses *Hadamard matrices* to produce a version of A that can be applied extremely quickly [AC09].

An important application of dimensionality reduction is in accelerating numerical algorithms on large matrices. See for example [Woo14a; DM17].

9.2 Gaussian random variables: an interface

Gaussian random variables are an *extremely convenient* class of random variables. To stress this point, rather than giving an explicit definition and proceeding with the mathematical analysis, we first outline (just a few of) the nice properties of Gaussian random variables, and put them to basic use. Later we will prove these properties, mostly by elementary calculus.

A Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ is parametrized by two parameters μ and σ^2. We write $x \sim \mathcal{N}(\mu, \sigma^2)$ to denote a real-valued random variable $x \in \mathbb{R}$ sampled by the (yet unspecified) Gaussian distribution. The paramters μ and σ^2 have simple interpretations.

Fact 9.2. *Let $x \sim \mathcal{N}(\mu, \sigma^2)$. Then the mean and variance of x are*

$$\mathbf{E}[x] = \mu \textit{ and } \operatorname{Var}[x] = \sigma^2.$$

We abbreviate $\mathcal{N} \overset{\text{def}}{=} \mathcal{N}(0, 1)$ for the special case of a Gaussian random variable with mean 0 and variance 1.

Some simple operations on Gaussian's produce new Gaussian's with their parameters naturally modified. First, scaling or shifting a Gaussian produces another Gaussian.

Fact 9.3. *Let $x \sim \mathcal{N}(0, \sigma^2)$ and let $\alpha \in \mathbb{R}$. Then $\alpha x \sim \mathcal{N}(0, \alpha^2\sigma^2)$ and $x + \alpha \sim \mathcal{N}(\alpha, \sigma^2)$.*

Second, adding two Gaussians produces another Gaussian with the means and variances (necessarily) added together.

Fact 9.4. *If $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, then $x_1 + x_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$.*

We also note that Gaussian's have nice exponential moments. Recall that exponential moments previously appeared when developing the Chernoff bound. Likewise, the following fact will eventually imply (below) that sums of Gaussians squared are well concentrated.

Fact 9.5. *Let $x \sim \mathcal{N}$ and $t < 1/2$. Then*

$$\mathbf{E}\left[e^{tx^2}\right] = \frac{1}{\sqrt{1 - 2t}}.$$

9.2.1 Concentration of length

We are also interested in ensembles of Gaussian random variables. For $k \in \mathbb{N}$, let $\mathcal{N}^k(\mu, \sigma^2)$ denote the distribution of k-dimensional vectors where each coordinate is a (μ, σ^2)-Gaussian. That is, when we write $x \in \mathcal{N}^k(\mu, \sigma^2)$, we mean that each $x_i \sim \mathcal{N}(\mu, \sigma^2)$, independently. Note that for $x \in \mathcal{N}^k(0, \sigma^2)$ has expected squared length

$$\mathbf{E}\left[\|x\|^2\right] = \sum_{i=1}^k \mathbf{E}\left[x_i^2\right] = \sum_{i=1}^k \mathrm{Var}[x_i] = k\sigma^2.$$

As a direct consequence of fact 9.5 above, the squared length $\|x\|^2$ of a Gaussian vector $x \sim \mathcal{N}^k$ will be extremely well concentrated, as follows.

Fact 9.6. *Let $x \sim \mathcal{N}^k(0, \sigma^2)$ be a Gaussian vector. Let $\alpha \geq 0$.*

1. *If $\alpha \leq 1$, then*

$$\mathbf{P}\left[\|x\|^2 \leq \alpha\, \mathbf{E}\left[\|x\|^2\right]\right] \leq \left(\alpha e^{1-\alpha}\right)^{k/2}.$$

2. If $\alpha \geq 1$, then

$$\mathbf{P}\Big[\|x\|^2 \geq \alpha\,\mathbf{E}\Big[\|x\|^2\Big]\Big] \leq \Big(\alpha e^{1-\alpha}\Big)^{k/2}.$$

Proof. Let us prove this fact because we only need the above facts to do so. Scaling x (and invoking fact 9.3), we can assume that $x \in \mathcal{N}^k$ and $\mu = \mathbf{E}\Big[\|x\|^2\Big] = k$.

Let $\alpha \in [0,1]$. For $t > 0$, we have

$$\begin{aligned}\mathbf{P}\Big[\|x\|^2 \leq \alpha k\Big] &= \mathbf{P}\Big[e^{-t\|x\|^2} \geq e^{-t\alpha k}\Big] \overset{\text{(a)}}{\leq} \mathbf{E}\Big[e^{-t\|x\|^2}\Big]e^{t\alpha k} \overset{\text{(b)}}{=} \left(\prod_{i=1}^{k}\mathbf{E}\Big[e^{-tx_i^2}\Big]\right)e^{t\alpha k} \\ &\overset{\text{(c)}}{=} \left(\frac{1}{1+2t}\right)^{k/2}\exp(\alpha tk) = \exp\!\left(k\left(\alpha t - \frac{1}{2}\ln(1+2t)\right)\right).\end{aligned}$$

(a) is by Markov's inequality. (b) is by independence of the x_i's (noting that $\|x\|^2 = \sum_i x_i^2$). (c) is by fact 9.5. The (exponent of the) RHS is minimized by

$$\alpha = \frac{1}{1+2t} \iff t = \frac{1-\alpha}{2\alpha}. \tag{9.1}$$

Plugging in t per (1) gives

$$\mathbf{P}\Big[\|x\|^2 \leq \alpha k\Big] \leq \Big(\alpha e^{1-\alpha}\Big)^{k/2},$$

as desired.

Now let $\alpha \geq 1$. For any $t \in (0,1/2)$, we have

$$\begin{aligned}\mathbf{P}\Big[\|x\|^2 \geq \alpha k\Big] &= \mathbf{P}\Big[e^{t\|x\|^2} \geq e^{t\alpha k}\Big] \overset{\text{(d)}}{=} \mathbf{E}\Big[e^{t\|x\|^2}e^{-t\alpha k}\Big] \\ &\overset{\text{(e)}}{\leq} \left(\frac{1}{1-2t}\right)^{k/2}e^{-\alpha tk} = \exp\!\left(-\frac{k}{2}(2\alpha t + \ln(1-2t))\right)\end{aligned}$$

by (d) Markov's inequality and (e) fact 9.5. The RHS is minimized at

$$\alpha = \frac{1}{1-2t} \iff t = \frac{\alpha-1}{2\alpha};$$

moreover, the RHS is in $(0,1/2)$ for all $\alpha > 1$. Plugging in, we have

$$\mathbf{P}\Big[\|x\|^2 \geq \alpha k\Big] \leq \Big(\alpha e^{1-\alpha}\Big)^{k/2},$$

as desired. □

An important case is where $\alpha = (1 \pm \epsilon)$ and $\epsilon > 0$ is close to 0. Then fact 9.6 implies the following.

Lemma 9.7. *Let $x \sim \mathcal{N}^k(0, \sigma^2)$ be a Gaussian vector. Let $\epsilon \in (0, 1]$. Then*

(i) $\mathbf{P}\Big[\|x\|^2 \leq (1 - \epsilon)\,\mathbf{E}\Big[\|x\|^2\Big]\Big] \leq e^{-c\epsilon^2 k/2}$ *for* $c = 1 - \ln(2) \approx .307$.

(ii) $\mathbf{P}\Big[\|x\|^2 \geq (1 + \epsilon)\,\mathbf{E}\Big[\|x\|^2\Big]\Big] \leq e^{-\epsilon^2 k/4}$.

Proof. By scaling, we can assume that $\sigma^2 = 1$ and $\|x\|^2 = 1$. We have

$$\mathbf{P}\Big[\|x\|^2 \geq (1 + \epsilon)k\Big] \overset{(a)}{\leq} \Big((1 + \epsilon)e^{-\epsilon}\Big)^{k/2} \overset{(b)}{\leq} e^{c\epsilon^2 k/2}.$$

Here (a) is by fact 9.6. (b) is because $1 + \epsilon \leq e^{\epsilon - c\epsilon^2}$ for $\epsilon \in [0, 1]$.

Likewise, we have

$$\mathbf{P}\Big[\|x\|^2 \leq (1 - \epsilon)k\Big] \overset{(c)}{\leq} ((1 - \epsilon)e^{\epsilon})^{k/2} \overset{(d)}{\leq} e^{-\epsilon^2 k/4}.$$

(c) is by fact 9.6. (d) is because $1 - \epsilon \leq e^{-\epsilon - \epsilon^2/2}$ for $\epsilon \in [0, 1]$. □

9.3 Random Projections

So far, we know that Gaussian random variables can be scaled and added together, and that the length of a squared Gaussian vector is well concentrated around its expectation. In fact this is all we need for the dimensionality reduction result mentioned above. The first lemma considers the projection of a single vector.

Lemma 9.8. *Let $A \sim \mathcal{N}^{k \times d}$ be a random matrix where each coordinate A_{ij} is an independently drawn sample from $\mathcal{N}$. Let $\epsilon > 0$ be sufficiently small. For any vector x,*

$$\mathbf{P}\bigg[(1 - \epsilon)\|x\|^2 \leq \frac{1}{k}\|Ax\|^2 \leq (1 + \epsilon)\|x\|^2\bigg] \geq 1 - 2e^{-k/8}.$$

Proof. Scaling if necessary, we can assume without loss of generality that $\|x\| = 1$. For $i \in [k]$, let $a_i = A^T e_i$ be the ith row of A. We have $a_i \sim \mathcal{N}^n$. Consider $\langle a_i, x\rangle = (Ax)_i$, as a random variable. By facts 9.3 and 9.4, $\langle a_i, x\rangle$ is a Gaussian random variable with mean 0 and variance

$$\sum_{j=1}^{n} \mathrm{Var}[x_j A_{ij}] = \sum_{j=1}^{n} x_j^2 = 1.$$

That is, $\langle a_i, x\rangle \sim \mathcal{N}$ for each i. In turn, we have $(Ax) \sim \mathcal{N}^k$. As a k-dimensional Gaussian vector, $\|Ax\|^2$ will be very well concentrated at its mean per Lemma 9.7. □

Consider Theorem 9.1 from the introduction, where we have a set of n points $P \subseteq \mathbb{R}^d$, and randomly project it into $\mathbb{R}^k$ with the linear function

$$f(x) = \frac{1}{\sqrt{k}} Ax,$$

where $k = O(\log(n)/\epsilon^2)$ and $A \sim \mathcal{N}^{k \times d}$. By Lemma 9.8, for each $x \in P$, we have

$$(1-\epsilon)\|x\|^2 \leq \|f(x)\|^2 \leq (1+\epsilon)\|x\|^2 \tag{9.2}$$

with probability of error (say) $\leq 1/n^{10}$. By the union bound, we have (2) for all x with probability of error $\leq 1/n^9$.

Theorem 9.1 also promised that all pairwise distances are preserved up to an $(1 \pm \epsilon)$-multiplicative factor. By *linearity* of f, we have

$$\|f(x) - f(y)\|^2 = \|f(x-y)\|^2$$

for any two points $x, y \in P$. We now argue, as before, that the lengths of the pairwise differences $x - y$ are all preserved with high probability.

9.4 Gaussians

Based on Theorem 9.1, a distribution satisfying facts 9.2–9.5 (from which all other facts and theorems are derived) may seem too good to be true. Let us now define this distribution formally and verify these simple facts.

The *Gaussian or normal distribution with mean $\mu \in \mathbb{R}$ and variance $\sigma^2 \geq 0$* is the real-valued random variable with density function

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{9.3}$$

(By Lemma 9.9 below, this random variable indeed has mean μ and variance σ^2.) We let $\mathcal{N}(\mu, \sigma^2)$ to denote the normal distribution with mean μ and variance σ^2, and write $X \sim \mathcal{N}(\mu, \sigma^2)$ to denote a random variable $X \in \mathbb{R}$ with distribution $\mathcal{N}(\mu, \sigma^2)$. A *normalized Gaussian* or *standard normal* random variable is a Gaussian random variable with mean 0 and variance 1. We abbreviate $\mathcal{N}(0,1)$ by $\mathcal{N}$. For $n \in \mathbb{N}$, we let $\mathcal{N}^n$ denote the joint distribution of n independent normalized Gaussian random variables.

9.4.1 Some preliminary calculus

Lemma 9.9. *Let $\mu \in \mathbb{R}$, $\sigma > 0$, and*

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}.$$

Then we have the following.

1. $\displaystyle\int_{-\infty}^{\infty} f(x) = 1.$
2. $\displaystyle\int_{-\infty}^{\infty} x f(x) = \mu.$
3. $\displaystyle\int_{-\infty}^{\infty} (x-\mu)^2 f(x) = \sigma^2.$

Proof. We consider the normalized case $\mu = 0$ and $\sigma = 1$. The general case follows by appropriate change of variables. We have

$$\begin{aligned}\left(\int_{-\infty}^{\infty} e^{-x^2/2}\,dx\right)^2 &= \int_{-\infty}^{\infty} e^{-(x^2+y^2)/2}\,dxdy = \int_0^{2\pi}\int_0^{\infty} re^{-r^2/2}\,drd\theta \\ &= 2\pi\int_0^{\infty} re^{-r^2/2}\,dr = 2\pi.\end{aligned}$$

Taking the square root of both sides gives the first claim. For the second claim, we have

$$\int_{-\infty}^{\infty} xe^{-x^2/2}\,dx = \left[e^{-x^2/2}\right]_{-\infty}^{+\infty} = 0.$$

For the third claim, we have

$$\int_{-\infty}^{\infty} x^2 e^{-x^2/2}\,dx \overset{\text{(a)}}{=} \left[-xe^{-x^2/2}\right]_{-\infty}^{\infty} + \int_{-\infty}^{\infty} e^{-x^2/2} = 1$$

by (a) integration by parts. □

Lemma 9.9 immediately implies both fact 9.2 and fact 9.3, which we restate for convenience.

Fact 9.2. *Let $x \sim \mathcal{N}(\mu, \sigma^2)$. Then the mean and variance of x are*

$$\mathbf{E}[x] = \mu \text{ and } \mathrm{Var}[x] = \sigma^2.$$

Fact 9.3. *Let $x \sim \mathcal{N}(0, \sigma^2)$ and let $\alpha \in \mathbb{R}$. Then $\alpha x \sim \mathcal{N}(0, \alpha^2\sigma^2)$ and $x + \alpha \sim \mathcal{N}(\alpha, \sigma^2)$.*

9.4.2 Rotational symmetry of Gaussian vectors

Let $X \sim \mathcal{N}^n$, and let $f(x)$ be the density function of x. The density function has the following compact form. The key feature is that the density at a point only depends on the squared length of the point. That is, it is *rotationally symmetric*.

Lemma 9.10. $f(x) = (2\pi)^{-n/2} e^{-\langle x,x \rangle/2}$.

Proof. Since each $x_i \sim \mathcal{N}$ independently, we have

$$f(x) = \prod_{i=1}^{n} (2\pi)^{-1/2} e^{-x_i^2/2} = (2\pi)^{-n/2} e^{-\langle x,x \rangle/2},$$

as desired. □

Lemma 9.11. *For any orthonormal matrix U, and random Gaussian vector x, $Ux \sim \mathcal{N}^n$.*

Proof. U induces a rotation, and the Gaussian is rotationally symmetric. For those who prefer explicit calculations, we have

$$f(Ux) \overset{\text{(a)}}{=} (2\pi)^{-n/2} e^{-\langle Ux,Ux \rangle/2} \overset{\text{(b)}}{=} (2\pi)^{-n/2} e^{-\langle x,x \rangle/2} \overset{\text{(c)}}{=} f(x),$$

where (a) is by Lemma 9.10, (b) is because $U^T U = I$, and (c) is by Lemma 9.10. □

Lemma 9.12. *Let $x \sim \mathcal{N}^n$ and $u \in \mathbb{R}^n$. Then $\langle u, x \rangle \sim \mathcal{N}\left(0, \|u\|^2\right)$.*

Proof. It suffices to assume u is a unit vector. By extending u to an orthonormal basis, let $u = U^T e_1$ for an orthogonal matrix U. Then

$$\langle u, x \rangle = \left\langle U^t e_1, x \right\rangle = \langle e_1, Ux \rangle = (Ux)_1 \overset{\text{(a)}}{\sim} \mathcal{N},$$

where (a) is by Lemma 9.11. □

Fact 9.4, which says that Gaussians sum nicely, now follows by combination of fact 9.3 and Lemma 9.12. We restate fact 9.4 for convenience and leave the proof to the reader.

Fact 9.4. *If $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, then $x_1 + x_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$.*

9.4.3 Moments of squared Gaussian random variables

The last fact to prove concerns the moment generating function of the square of a Gaussian random variable. Recall that amplifying the following bound leads to the concentration of length of high-dimensional Gaussian vectors, which in turn, allows us to obliviously embed high-dimensional data in Theorem 9.1.

Fact 9.5. *Let $x \sim \mathcal{N}$ and $t < 1/2$. Then*

$$\mathbf{E}\Big[e^{tx^2}\Big] = \frac{1}{\sqrt{1-2t}}.$$

Proof. We have

$$\begin{aligned}\mathbf{E}\Big[e^{tx^2}\Big] &= \int_s e^{ts^2}\,\mathbf{P}[x=s] \overset{\text{(a)}}{=} \frac{1}{\sqrt{2\pi}}\int_s e^{(2t-1)s^2/2} \\ &= \frac{1}{\sqrt{2\pi}}\int_s e^{-s^2/2\sigma^2} \qquad \text{for } \sigma = 1/\sqrt{1-2t} \\ &\overset{\text{(b)}}{=} \frac{1}{\sqrt{1-2t}}.\end{aligned}$$

Here (a) plugs in the density function from equation (9.3). (b) is by Lemma 9.9.1 w/r/t the density function for $\mathcal{N}(0,\sigma^2)$. □

9.5 Additional notes and materials

Lecture materials. Click on the links below for the following files:
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

9.6 Exercises

Exercise 9.1. Using only fact 9.3, show that for $x \sim \mathcal{N}(\mu, \sigma^2)$ and $\alpha \in \mathbb{R}$,

$$\alpha x \sim \mathcal{N}\Big(\alpha\mu, \alpha^2\sigma^2\Big).$$

Exercise 9.2. Show that there exists universal constants $c_1, c_2 > 0$ such that for all x with $|x| \leq c_1$,

$$1 + x \leq e^{x - c_2 x^2}.$$

(In other words, you can choose whatever constants c_1 and c_2 are convenient to you.)

Exercise 9.3. Let $P \subseteq \mathbb{R}^d$ be a set of n points. Let $f : \mathbb{R}^d \to \mathbb{R}^k$ be a random projection with $k = O(\log(n)/\epsilon^2)$ (per Theorem 9.1). Recall that with high probability (say, $\geq 1 - 1/n^4$), we have

$$(1-\epsilon)\|x\|^2 \leq \|f(x)\|^2 \leq (1+\epsilon)\|x\|^2$$

for all $x \in P$, and we also have

$$(1-\epsilon)\|x-y\|^2 \leq \|f(x) - f(y)\|^2 \leq (1+\epsilon)\|x-y\|^2$$

as well as

$$(1-\epsilon)\|x+y\|^2 \leq \|f(x) + f(y)\|^2 \leq (1+\epsilon)\|x+y\|^2$$

for all $x, y \in P$. Show that with high probability, we also have

$$|\langle f(x), f(y)\rangle - \langle x, y\rangle| \leq \epsilon\|x\|\|y\|$$

for all $x, y \in P$.[1]

Exercise 9.4. Let $P \in \mathbb{R}^{\ell \times n}$ be the random projection function as described in Theorem 9.1, for a parameter ℓ to be determined. We want to argue that for $\ell = O(k/\epsilon^2)$, P approximately preserves all of the vectors of a fixed subspace U of dimension k with high probability (in k).

[1] It might by helpful to work through the special case where $\|x\| = \|y\| = 1$. Showing $O(\epsilon\|x\|\|y\|)$ error is fine; a proper $\epsilon\|x\|\|y\|$ then follows from dividing ϵ by a constant factor.

To express this more formally, let U be a fixed (but unknown) subspace of $\mathbb{R}^n$ of dimension k. We claim that, with probability at least $1 - e^{-O(k)}$, we have

$$(1-\epsilon)\|x\|^2 \leq \|Px\|^2 \leq (1+\epsilon)\|x\|^2 \text{ for all } x \in U \text{ (simultaneously).} \tag{9.4}$$

Note that the algorithm does not know U; for this reason, P is called an $(1 \pm \epsilon)$-approximate *oblivious subspace embedding.* Oblivious subspace embeddings are useful for developing faster approximation algorithms in numerical linear algebra.

In this exercise we prove that P is an oblivious subspace embeddings with high probability. Let U be a subspace of dimension k. Let $\mathbb{S}^k$ be the unit sphere in the k-dimensional subspace we want to preserve. Since the requirements of (C.1) are scale invariant, it suffices to establish (C.1) for all points in $\mathbb{S}^k$.

Our argument makes use of a geometric technique called *ϵ-nets.* For a set S, an ϵ-net is a set N such that for every point $s \in S$, there is a point $x \in N$ such that $\|x - s\| \leq \epsilon$.[2] We need to show that there exists a relatively small ϵ-net for $\mathbb{S}^k$.

1. Let $N \subseteq \mathbb{S}^k$ be a maximal set of points such that any two points in N have distance at least ϵ. Show that N is an ϵ-net, and that N has at most $(1 + 2/\epsilon)^k$ points.[3]

Let N be an $(1/2)$-net of $\mathbb{S}^k$ with at most 5^k points. Next we establish that we preserve the length of all points in N is preserved with high probability.

2. Show that with probability $1 - e^{-O(k)}$, we have $\left|\|Px\|^2 - 1\right| \leq \epsilon$ for all $x \in N$, and $|\langle Px, Py\rangle - \langle x, y\rangle| \leq \epsilon$ for all $x, y \in N$.

So now we know that we preserve all points of N with high probability. We want to argue that this suffices to preserve all the vectors.

3. Prove that for any unit vector $x \in \mathbb{S}^k$, one can write $x = x_0 + x_1 + x_2 + \cdots$ such that for all i:
   (a) $\|x_i\| \leq 2^{-i}$.
   (b) $x_i/\|x_i\| \in N$.[4]

[2]This is similar but different from the ϵ-nets in Chapter 13.

[3]*Hint:* N packs $|N|$ interior-disjoint k-dimensional balls of radius $\epsilon/2$ into an k-dimensional ball of radius $1 + \epsilon/2$. It is helpful to know that the volume of an n-dimensional ball of radius r is $c_n r^n$ for a parameter $c_n > 0$ depending on n.

[4]Hint: Choose x_0 to be the closet point in N to x. Observe that $\|x - x_0\| \leq 1/2$ because N is a (1/2)-net. It remains to express $x - x_0$ as the sum $x_1 + x_2 + \cdots$. How might you choose x_1?

Now use the representation above to prove that P is an oblivious subspace embedding.

4. Prove that (C.1) holds with probability at least $1 - e^{-O(k)}$.

The high-level takeaway from the proof is that if you can embed an ϵ-net of the unit sphere for constant ϵ, then you automatically embed the entire subspace.

Chapter 10

Locality sensitive hashing and approximate nearest neighbors

10.1 Nearest neighbor search

In *similarity search*, we want to preprocess a large database of items such that, given a query in the form of another item, we can quickly retrieve the "most similar" item in our collection by some metric. The "most similar" item is sometimes called the *nearest neighbor*. Of course, one could directly compare the queried item to every element in the database and output the most similar. The goal is to structure the data to serve the queries in sublinear time.

As a simple warmup example, suppose we want to implement similarity search 1 dimension over a database of n real numbers. Each query is defined by a real value q, and we want to retrieve the closest number in the database to q.

1-dimensional similarity search is easy. We first sort our collection of numbers and store them in an array. Given a query $q \in \mathbb{R}$, we run binary search on the array to find the first numbers smaller and bigger than q. We return the closer of the two. This takes $O(\log n)$ time. The above approaches extends to low-dimensional data sets, via *quadtrees* and multi-dimensional *range trees*. Both of these approaches scale exponentially in the dimension d. If the data lives in very few dimensions - like 3 dimensions, as in many physical situations - this is still very good.

As previously discussed, there are many natural settings where the data is represented geometrically as *high-dimensional* vectors. We are interested in nearest neighbor search over these large vectors. The input consists of n points $P \subset \mathbb{R}^d$ that we can preprocess to build some kind of data structure. Each query is specified by another point $z \in \mathbb{R}^d$, and the goal is to output the point $x \in P$ closest to z by some metric. We will consider two metrics.

1. The *Euclidean distance*, $\|x - z\|$.

2. The *angle* between x and z (as a real value between 0 and π).

If all the vectors in P are normalized to have the same length, then the nearest neighbor in Euclidean distance and angle are the same. However, we will consider *approximations* for this problem, in which case there is a difference between the two metrics. We consider angular nearest neighbors in Section 10.2 and Euclidean nearest neighbors in Section 10.3.

A natural idea, at least for Euclidean distance, is to use the dimensionality reduction techniques previously discussed. Recall that by simply projecting onto vectors of independent Gaussian entries, an n-point data can be embedded into $O(\log(n)/\epsilon^2)$ dimensions while preserving all pairwise distances up to an $(1 \pm \epsilon)$-multiplicative factor. Perhaps one insert these lower-dimensional embeddings into geometric data structures like quadtrees. Unfortunately, the exponential dependence on the dimension incurred by these data structures means that even $O(\log n)$ dimensions – which was fairly small by our standards – still requires $n^{O(1)}$ time and space.

From nearest neighbor to "close enough". One can reduce nearest neighbors to distance-based membership queries. An approximate distance-based membership data structure takes as input a collection of points P, and is parametrized by a radius $r > 0$ and an approximation factor $\sigma \geq 1$. Given a query point q, the goal is to either:

1. Output a point $p \in P$ at distance $\leq \sigma r$; or
2. Declare that all points have distance $\geq r$.

We can reduce σ-approximate nearest neighbor queries to a series of (r, σ)-membership queries as follows. Suppose all possible distances of points between P are in the range $[\Delta_{\min}, \Delta_{\max}]$. For $i \in \mathbb{Z}_{\geq 0}$, let

$$r_i = \sigma^i.$$

We create an (r_i, σ)-data structure for every radius r_i with $\Delta_{\min}/2 \leq r_i \leq \text{poly}(n)\Delta_{\max}$. This creates $\log_\sigma(\text{poly}(n)\Delta_{\max}/\Delta_{\min})$ total membership data structures.

Now, suppose we want to find a σ-approximate nearest neighbor to a query point q. We identify, via binary search, the smallest radius r_i for which the corresponding membership query returns a point p_i. Since the membership query r_{i-1} failed, we know that r_i is at most a σ-factor greater than the minimum distance between q and P, and the returned point p_i is a σ-approximate data structure.

Note that via binary search, we only need

$$\log(\log_\sigma(\text{poly}(n)\Delta_{\max}/\Delta_{\min})) = O(\log\log(n\Delta_{\max}/\Delta_{\min}) + \log(\sigma))$$

membership queries to identify r_i.

We also observe that it suffices, up to logarithmic factors, to succeed with constant probability. We can amplify to high probability with repetition.

LSH: when hash collisions are good. The algorithms we develop use hashing in a counterintuitive way. Previously, in applications such as heavy hitters and hash tables, hashing was used to randomly *spread out* the elements and reduce collisions. Here, we will design hash functions where closer points are *more likely to collide*. This technique is called *locality sensitive hashing* (LSH) [IM98].[1] The high level strategy is to build a hash table over the input set P using locality sensitive hash functions. Given a query point z, we hash z and hope to find the nearest neighbor in the same hash bucket.

Such an algorithm may only succeed with limited probability. We amplify by building many such hash tables independently. On a query point z, we hash z into all of the hash tables, and return the first point we find that is close enough.

10.2 LSH for angles

Suppose the point set P lies on the hypersphere $\mathbb{S}^{d-1} = \left\{x \in \mathbb{R}^d : \|x\| = 1\right\}$. A popular measure of distance on $\mathbb{S}^{d-1}$ is the *angle* between points. For two points x and y, let $\angle(x, y) \in [0, \pi]$ denote the angle between the points x and y between 0 and π. We may assume that all the points in our data set P, as well as any query point, is normalized to lie on $\mathbb{S}^{d-1}$. We describe an LSH scheme due to [Cha02].

Let $\theta \in [0, \pi]$ be a fixed angle. Our goal is to implement a 2-approximate membership query for angular distance θ. Formally, given a query point y, we must either:

(i) Output a point $x \in P$ with $\angle(x, y) \leq 2\theta$, or

(ii) Decide that there are no points $x \in P$ with $\angle(x, y) \leq \theta$.

Let $k \in \mathbb{N}$ be a parameter TBD. We will generate a k-ary hash function $h : \mathbb{S}^{d-1} \to \{-1, 1\}^k$ where each coordinate is generated by splitting $\mathbb{S}^{d-1}$ in half along a random hyperplane. For each coordinate $i \in [k]$, let $g_i \sim \mathcal{N}^d$ be a random Gaussian vector. We define a hash function $h : \mathbb{S}^{d-1} \to \{-1, 1\}^k$ by

$$h_i(x) = \operatorname{sign}(\langle g_i, x \rangle)$$

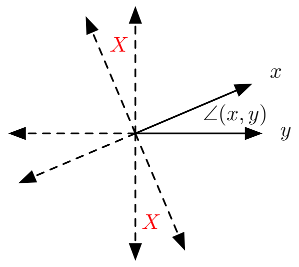
for each coordinate i.

At a high level, we want to argue that collisions in h are correlated with angle. We leverage the following fact, which is entirely based on the rotational invariance of $\mathcal{N}^d$.

[1] Here the author would like to propose the term *clashing*.

Lemma 10.1. *Let $x, y \in \mathbb{S}^{d-1}$, and let $g \sim \mathcal{N}^d$. Then*

$$\mathbf{P}[\operatorname{sign}(\langle g, x\rangle) \neq \operatorname{sign}(\langle g, y\rangle)] = \frac{\angle(x,y)}{\pi}.$$



Proof. Recall that $\mathcal{N}^d$ is *rotationally symmetric*. Rotating, we may assuming that x and y are spanned by the first two coordinates, and consider the simpler (to visualize) setting where $x, y \in \mathbb{R}^2$ and $g \sim \mathcal{N}^2$, as on the right.

In $\mathbb{R}^2$, g makes an angle (with the x-axis) between 0 and 2π. By symmetry, it is equally likely to take any particular angle. There are two regions of angle $\angle(x,y)$ in which $\operatorname{sign}(\langle g, x\rangle) \neq \operatorname{sign}(\langle g, y\rangle)$, here marked with a red X. The odds of g landing in these regions is $\angle(x,y)/\pi$. ☐

We set the number of hash coordinates to $k = \frac{\pi \log(n)}{2\theta}$. If $\angle(x,y) < \theta$, then

$$\mathbf{P}[h(x) = h(y)] = \left(1 - \frac{\angle(x,y)}{\pi}\right)^k \approx e^{-\frac{\angle(x,y)}{\pi}k} = e^{-\log(n)/2} = \frac{1}{\sqrt{n}}.$$

On the other hand, if $\angle(x,y) > 2\theta$, then

$$\mathbf{P}[h(x) = h(y)] = \left(1 - \frac{\angle(x,y)}{\pi}\right)^k \leq e^{-\angle(x,y)k/\pi} \leq e^{-\log(n)} = \frac{1}{n}.$$

Thus, when querying a point y:

1. If there is a point $x \in P$ with $\angle(x,y) \leq \theta$, then it collide with y with probability (approximately) $\geq 1/\sqrt{n}$.

2. We expect to collide with at most 1 point $x \in P$ such that $\angle(x,y) \geq 2\theta$.

To amplify our success rate, we repeat the experiment $O(\sqrt{n}\log(n))$ times. Then, if there is a θ-close point, we will collid with it with high probability. The query time is $O(d\sqrt{n}\log(n))$ in expectation because we expect 1 "junk" neighbor per hash table on average.

Theorem 10.2. *With $O\big(dn^{3/2}\operatorname{polylog}(n)\big)$ preprocessing time and space, one can query for 2-approximate nearest neighbors w/r/t angular distance in $O(d\sqrt{n}\operatorname{polylog}(n))$ expected randomized time and with high probability.*

10.3 LSH for Euclidean distance

We now consider σ-approximate nearest neighbors in Euclidean metrics, for fixed $\sigma > 1$. Again it suffices to consider fixed-radius approximate membership queries. Given a fixed radius r and a set of n points $P \subseteq \mathbb{R}^d$, we want to preprocess P to be able to quickly serve the following query: given a query point q, either:

(i) Return a point $p \in P$ with $\|p - q\| \leq \sigma r$; or

(ii) Declare that there are no points $p \in P$ with $\|p - q\| \leq r$.

Our goal is to develop a locality-sensitive hash function $h : \mathbb{R}^d \to \mathbb{Z}^k$ for Euclidean distance. We will design a scale-invariant function h such that the collision probobilities of two points $x, y \in \mathbb{R}^d$ depend only on the ratio $\|x - y\|/r$ to r. For ease of notation, we rescale and assume that $r = 1$. Thus, given a query point $q \in \mathbb{R}^d$, we want to:

(i) Return a point $p \in P$ with $\|p - q\| \leq \sigma$; or

(ii) Declare that there are no points $p \in P$ with $\|p - q\| \leq 1$.

10.3.1 Random line embeddings and buckets

We first define a hash function $h : \mathbb{R}^d \to \mathbb{Z}$ that outputs a single integer. Later we will consider a k-coordinate hash where each coordinate is an independent copy of this single-coordinate hash.

To build some intuition, consider the 1-dimensional case $d = 1$. As mentioned earlier, we can build a binary search tree over P and serve queries in $O(\log n)$ time. Of course this answers the problem exactly. To do even faster, we can try to take advantage of the margin of error allowed in our approximation queries.

Consider the following hash code $h : \mathbb{R} \to \mathbb{Z}$:

$$h(x) = \left\lfloor \frac{x + \alpha}{\sigma} \right\rfloor,$$

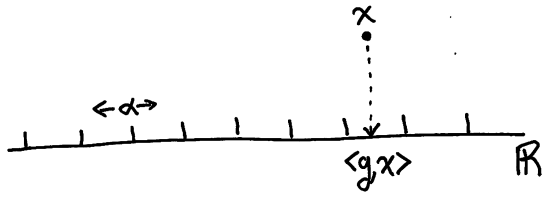
where $\alpha \in [0, 1]$ is drawn uniformly at random. $h(x)$ is splitting $\mathbb{R}$ into intervals of length σ, randomly translating the start of each interval.

Now, if two points $x, y \in \mathbb{R}$ have distance $|x - y| > \sigma$, when they will never collide. On the other hand, if $x, y \in \mathbb{R}$ have distance $|x - y|$, then they have a $1/\sigma$ chance of being hashed together. E.g., for $\sigma = 2$, we have a 50% chance of x and y colliding.

We transfer this "random bucket" approach to $\mathbb{R}^d$ by randomly projecting onto $\mathbb{R}$. We define $h : \mathbb{R}^d \to \mathbb{Z}$ by the function

$$h(x) = \lfloor \langle g, x \rangle + \alpha \rfloor,$$

where $g \sim \mathcal{N}^d$ and $\alpha \in [0,1]$ uniformly at random. Geometrically, $h(x)$ randomly projects x onto a line, and then bucket the points in intervals of length 1. The random value $\alpha \in [0,1]$ translates the buckets randomly.



For $x, y \in \mathbb{R}^d$, we want to bound the probability of a hash collision $h(x) = h(y)$. The first lemma conditions on g and analyzes the effect of the random translation α.

Lemma 10.3. *Let $x, y \in \mathbb{R}^d$.*

$$\mathbf{P}[h(x) = h(y) \mid g] = \max\{0, 1 - |\langle x - y, g\rangle|\}.$$

Proof. Once the Gaussian is fixed, so are the coordinates $\langle g, x\rangle$ and $\langle g, y\rangle$ on the line. We have $h(x) \neq h(y)$ iff a randomly shifted divider (determined by $\alpha \in [0,1]$) falls between $\langle g, x\rangle$ and $\langle g, y\rangle$. If $|\langle g, x\rangle - \langle g, y\rangle| \geq 1$, the divider always splits x and y. this always happens. For $|\langle g, x\rangle - \langle g, y\rangle| < 1$, x and y are split with probability $|\langle g, x\rangle - \langle g, y\rangle|$. □

Now we analyze the full probability of a hash collision.

Lemma 10.4. *Let $x, y \in \mathbb{R}^d$, and let $f(t)$ be the density function of the standard Gaussian $\mathcal{N}$.*

$$\mathbf{P}[h(x) = h(y)] = 2\frac{t}{\|x-y\|}\int_0^1 f\left(\frac{t}{\|x-y\|}\right)(1-t)$$

Proof. Recall that $\langle x - y, g\rangle \sim \mathcal{N}\left(0, \|x-y\|^2\right)$. In particular, $\langle x - y, g\rangle$ has density function $f(t/\|x-y\|)/\|x-y\|$, and $|\langle x - y, g\rangle|$ has density function $2f(\|x-y\|t)$. We have

$$\begin{aligned}\mathbf{P}[h(x) = h(y)] &= \frac{2}{\|x-y\|}\int_0^1 \mathbf{P}[h(x) = h(y) \mid |\langle g, x-y\rangle| = t] f(\|x-y\|t)\,dt \\ &= \frac{2}{\|x-y\|}\int_0^1 (1-t) f(\|x-y\|t)\,dt,\end{aligned}$$

as desired. □

Remark 10.5. For $t \in [0,1]$, $g(\sigma) = f(t/\sigma)/\sigma$ is minimized by $\sigma = 1$.

Lemma 10.4 gives the exact probability of a collision of two points x and y as a function of the distance between $\|x - y\|$. Let us compare the probabilities of a "close" pair of points, with $\|x - y\| \leq 1$, and a far pair of points, with $\|x - y\| \geq \sigma$. Let

$$p = 2 \int_0^1 (1-t) f(t)\, dt$$

be a lower bound on the collision probability when $\|x - y\| \leq 1$. Let

$$q = \frac{2}{\sigma} \int_0^1 (1-t) f(t/\sigma)\, dt$$

be an upper bound on the collision probability when $\|x - y\| \geq \sigma$. We note that p is a fixed constant, about .368746....

Since q is decreasing in σ, $p > q$. That is, close points are more likely to collide then far points. To what extent? It turns out that the gap is not big enough to use h directly as an LSH function. But we can remedy this by amplification, as follows.

10.3.2 Amplifying the gap

Let $k \in \mathbb{N}$ be a parameter TBD. We define a hash function

$$h : \mathbb{R}^d \to \mathbb{R}^k$$

by defining each coordinate $h_i(x)$ according to the single-coordinate hash function in Section 10.3.1; namely, as

$$h_i(x) = \lfloor \langle g_i, x \rangle + \alpha_i \rfloor$$

where $g_i \sim \mathcal{N}$ and $\alpha_i \in [0,1]$ uniformly at random.

For any two points x and y, by Lemma 10.4, we have

$$\mathbf{P}[h(x) = h(y)] = \left(\frac{2}{\|x - y\|} \int_0^1 (1-t) f\left(\frac{t}{\|x - y\|} \right) dt \right)^k.$$

In particular, recalling the values of p and q as above, we have

$$\mathbf{P}[h(x) = h(y)] \geq p^k \text{ when } \|x - y\| \leq 1$$

and

$$\mathbf{P}[h(x) = h(y)] \leq q^k \text{ when } \|x - y\| \geq \sigma.$$

Now, increasing k decreases both p^k and q^k, which is both good and bad. As p^k decreases, so do the odds of finding a near neighbor when we hash. We will need to rebuild the data structure $\ell = 1/p^k$ times to be able to find a good neighbor with constant probability, which is expensive.[2] On the other hand, as q^k decreases, the number of hash collisions with bad points also reduces. This is good algorithm pays a running time proportional to the total number of bad collisions as it scans the hash bucket. Overall, the ratio $(q/p)^k$ decreases, so to some extent, k is useful.

Ultimately, the quantity we want to minimize is

$$k\left(\frac{1}{p}\right)^k + \left(\frac{q}{p}\right)^k n. \tag{10.1}$$

The $(1/p)^k$ term represents having to compute $\ell = (1/p)^k$ hash codes. The $\left(\frac{q}{p}\right)^k n$ represents the expected number of hash collisions with bad elements across all ℓ instances. The full running time is the above quantity times d, since we need $O(d)$ time to hash a query point or compute the distance between two points.

We can rewrite (10.1) as

$$\left(\frac{1}{p}\right)^k \Big(k + q^k n\Big).$$

Let $k = \log(n)/\log(1/q) = \log(1/n)/\log(q)$. Then $q^k n = 1$, and

$$k\left(\frac{1}{p}\right)^k = \frac{\log(n)}{\log(1/q)} n^{\log(1/p)/\log(1/q)}.$$

Consider the exponent $\log(1/p)/\log(1/q)$: since $p > q$, this quantity is less then 1.

Theorem 10.6. *One can compute a σ-approximate nearest neighbor w/r/t Euclidean distance with high probability in $\tilde{O}\Big(dn^{\rho(\sigma)}\Big)$ randomized time, where*

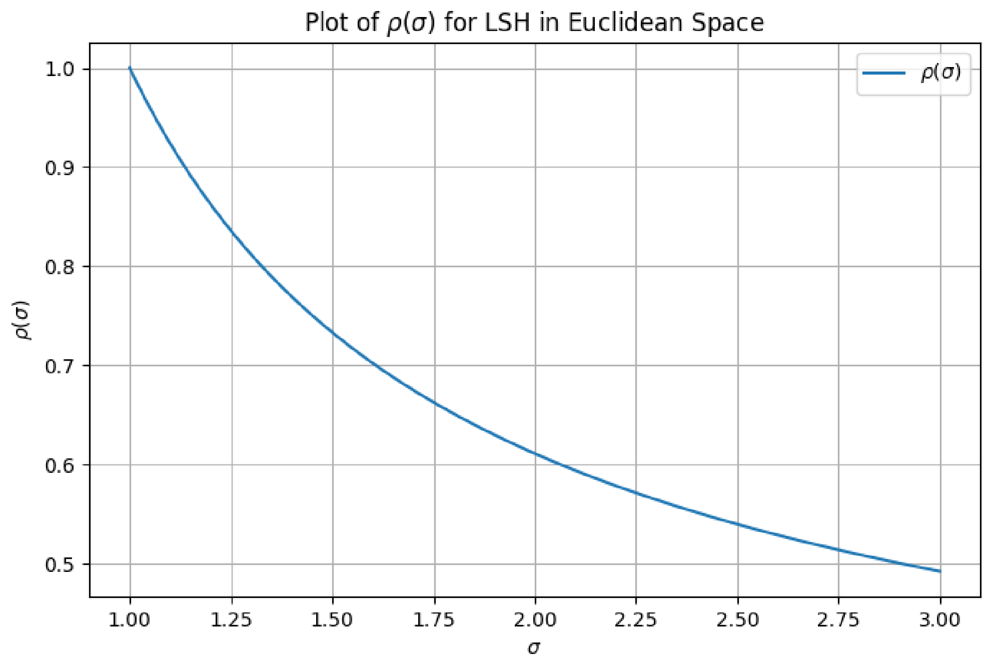
$$\rho(\sigma) \overset{\text{def}}{=} \frac{\log(1/p)}{\log(1/q)}, \qquad p \overset{\text{def}}{=} 2\int_0^1 (1-t) f(t)\, dt, \qquad q = \frac{2}{\sigma}\int_0^1 (1-t) f(t/\sigma)\, dt,$$

and $f(t) = e^{-t^2/2}/\sqrt{2\pi}$ is the density function of the standard Gaussian.

[2] Indeed, the probability of failing to find the good neighbor in each of ℓ constructions, each of which has k hash coordinates, is

$$\left(1 - p^k\right)^\ell \approx e^{-p^k \ell}.$$

What is $\rho(\sigma)$? Ask the computer:



10.4 Additional notes and materials

Lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

10.5 Exercises

Exercise 10.1. In this exercise, we will develop a 2-approximate LSH scheme for bit strings $s \in \{0,1\}^d$ of a fixed length d with respect to *Hamming distance*. The Hamming distance between two strings $s, t \in \{0,1\}^d$ this fraction of coordinates in which they differ:

$$\text{Hamming}(s,t) = \frac{|\{i \in [d] : s_i \neq t_i\}|}{d}.$$

Of course one can treat bit strings as vectors in $\mathbb{R}^d$ where the Hamming distance coincides with the Euclidean distance squared. Here we explore an alternative approach.

1. Consider the randomly constructed hash function $h : \{0,1\}^d \to \{0,1\}$ defined by

   $$h(x) = x_i,$$

   where $i \in [d]$ is sampled uniformly at random. For two points $s, t \in \{0,1\}^d$, what is $\mathbf{P}[h(s) = h(t)]$, as a function of the Hamming distance between s and t?

2. Fix a target distance $r \in [0,1]$. Construct a data structure that over a set P of n strings $\{0,1\}^d$ to answer the following query with high probability.

   Given a query point $s \in \{0,1\}^d$, either return a point $x \in P$ with Hamming distance $\leq 2r$ from s, or declare that there are no points within Hamming distance r from s.

   In addition to describing the algorithm, one should analyze the preprocessing time and space, the query time, and the probability of correctness.

3. Briefly describe how to use the above data structure to efficiently find 2-approximate nearest neighbors with respect to Hamming distance (with high probability). Analyze your running time.

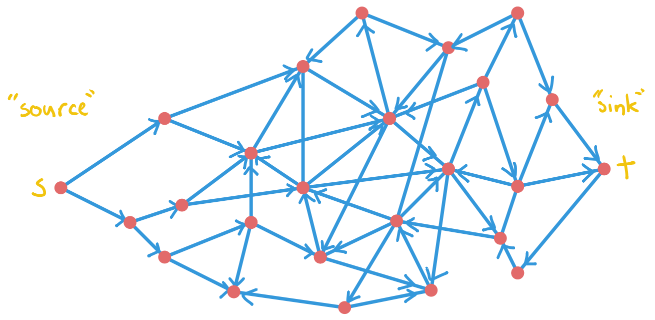
Chapter 11

L_1-metric embeddings and sparsest cut

11.1 LP duality, line embeddings, and max-flow min-cut

The reader may (should) have seen max flow, the max-flow min-cut theorem, and some algorithms for max flow in an introductory class on algorithms. Here we present these results from a different perspective that primes us for the techniques used to approximate the sparsest cut problem.

11.1.1 Packing and covering paths



Let $G = (V, E)$ be a directed graph[1], and let $s, t \in V$ be two distinct vertices. We call s the *source* and t the *sink*. A *path packing* is a collection of edge disjoint paths. An (s,t)-path packing is a path packing of (s,t)-paths. The *maximum* (s,t)*-path packing* problem[2] is to

find a maximum cardinality packing of (s,t)*-paths.*

An (s,t)*-cut* is a set of edges whose removal disconnects s from t. The *minimum* (s,t)*-cut problem* is to

[1] G is allowed to be a *multi-graph*, with multiple copies of the same edge.

[2] Better known as the *uncapacitated maximum* (s,t)*-flow problem* for reasons we discuss later.

find the minimum cardinality (s,t)-cut.

Both can be understood as a generalization of reachability. Reachability is concerned with whether there is a single connection from s to t. Both the maximum flow and minimum cut problems measure the *strength* of the connection from s to t.

We are interested in these questions both algorithmically and (graph-)structurally. Algorithmically the problems are highly non-trivial, as there are exponentially many possible paths from s to t to take into account. It is not obvious that there is a polynomial time algorithm for either problem.

11.1.2 Duality.

The (s,t)-path packing and (s,t)-cut problem are *dual* packing and covering problems, in the following sense. We are selecting paths that "pack in" to the edges of the graph – each path uses up all of its edges. Conversely, an (s,t)-cut must contain at least one edge from every (s,t)-path. That is, we are trying to "cover" the paths with edges, where we interpret each edge as a set that covers all the (s,t)-paths that contain that edge. In short:

paths pack into edges, and edges cover paths.

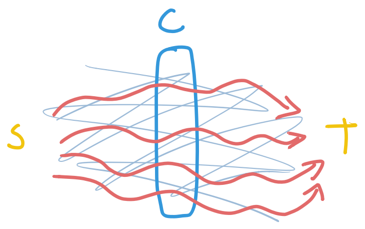
As with any packing and covering problem, we have the following inequality.

$$(\text{max } (s,t)\text{-path packing}) \leq (\text{min } (s,t)\text{-cut}). \tag{11.1}$$

Indeed, let P denote a path packing, and let $C \subseteq E$ be an (s,t)-cut. We have

$$|P| = \sum_{p \in P} 1 \overset{\text{(a)}}{\leq} \sum_{p \in P} |p \cap C| \overset{\text{(b)}}{\leq} |C|.$$

Above, we treat each path $p \in P$ as a subset of edges. (a) is because, as an (s,t)-cut, C contains at least one edge from every (s,t)-path. (b) is because the paths $p \in P$ are edge disjoint, so the sets $p \cap C$ over $p \in P$ are also disjoint. The following is a conceptual sketch of our argument.



The inequality (11.1) inspires some basic questions. Is the inequality ever equal? Is the inequality ever strict?

We now introduce some variations of the (s,t)-paths and cut problems.

11.1.3 Capacities and costs.

A natural generalization of disjoint (s,t)-paths allows edges to be reused, and extends the input to include *edge capacities* $c : E \to \mathbb{R}_{>0}$ that set a numerical limit on how many times each edge can be used. For example, if an edge e has $c(e) = 2$, then this implies we are allowed to use e *twice* (i.e., as if there are two copies of e). The same path is allowed to be selected multiple times. Formally, the problem becomes:

> *Find a collection of paths P of maximum cardinality $|P|$ such that each edge e is contained in at most $c(e)$ paths.*

We denote the problem formalized above as (Max-Paths). Abusing notation, we will also let (Max-Paths) denote the optimum objective value of the (Max-Paths) problem.

Introducing edge capacities makes it even more challenging to find a polynomial time algorithm. Before, without edge capacities, it is clear that the optimum solution has a polynomial number of paths since each path must use up at least one edge. Once we introduce capacities – which may be large numbers expressed in a logarithmic number of bits – we can no longer assume that the maximum path packing has polynomial size!

Flip back to the dual (s,t)-cut problem. A natural generalization introduces positive *edge costs* $c : E \to \mathbb{R}_{>0}$; the problem becomes:

$$\textit{Find an } (s,t)\textit{-cut } C \subseteq E \textit{ of minimum cost } \sum_{e \in C} c(e). \tag{Min-Cut}$$

The minimum cardinality cut problem is equivalent to the minimum cost cut problem with uniform costs ($c(e) = 1$ for all e). In contrast to edge capacities, edge costs do not invoke the risk of the optimum solution no longer being compact.

As with (Max-Flow), we write (Min-Cut) to denote both the problem described above and the optimum value of that problem.

When the capacities and costs are based on the same vector c, then (Max-Paths) and (Min-Cut) are dual[3] to one another. In particular one can show that (Max-Paths) $\leq$ (Min-Cut) by a similar argument as before in the uncapacitated/uniform-cost setting. We leave the proof to the reader.

Lemma 11.1. *Let $G = (V,E)$ be a directed graph and $s,t \in V$. Let $c : E \to \mathbb{R}_{>0}$ be a fixed set of capacities / costs. Then*

$$\text{(Max-Paths)} \leq \text{(Min-Cut)}.$$

[3]We will formalize the definition of dual soon.

11.1.4 Fractionally packing and covering paths.

Another variation of the above problems allow for *fractional solutions*. Consider first path packings. Let $\mathcal{P}_{s,t}$ denote the family of all (s,t)-paths. A *fractional collection of paths* is an assignment $x : \mathcal{P}_{s,t} \to \mathbb{R}_{\geq 0}$ giving nonnegative weight to every path. A *fractional path packing* is a fractional collection of paths x that satisfies the capacity constraint $c(e)$ for each edge e in the following quantitative sense:

$$\sum_{p \ni e} x_p \leq c(e).$$

(Here "$p \ni e$" is summing over all $p \in \mathcal{P}_{s,t}$ such that $e \in p$.) Subject to this constraint, the goal is to find the fractional path packing of maximum total quantity,

$$\sum_{p \in \mathcal{P}_{s,t}} x_p.$$

Putting it all together, the entire fractional path packing problem is given by:

$$\begin{aligned} \text{maximize } & \sum_{p \in \mathcal{P}_{s,t}} x_p \text{ over } x : \mathcal{P}_{s,t} \to \mathbb{R}_{\geq 0} \\ \text{s.t. } & \sum_{p \ni e} x_p \leq c(e) \text{ for all } e \in E. \end{aligned} \tag{Max-Flow}$$

Note that the objective ($\sum_p x_p$) is a linear function of x, and the constraints ($x_p \geq 0$, $\sum_{p \ni e} x_p \leq c(e)$...) are linear inequalities. This feature is very important and we will return to it in a moment.

Due to the continuous nature of the fractional path packing problems – imagine x_p units of water flowing along the path p – it is commonly referred to as the *maximum flow* problem. For the rest of this section, we let (Max-Flow) refer to the maximum flow problem formulated above. Abusing notation, we will also let (Max-Flow) refer to the optimum value of (the problem) (Max-Flow). We also note that the (discrete) path packing is also called *integral maximum flow*.

Compare max flow with the (integer) path packing problem discussed above, where we can only select integer multiples of paths. Clearly, any integer solution is a feasible solution to the fractional version. For this reason, the fractional path packing problem is called a *relaxation* of the integer path packing problem – every feasible solution to the latter is feasible in the former. As a relaxation of a maximization problem, we always have (Max-Flow) $\geq$ (Max-Paths).

We can apply the same fractional perspective to the minimum cost (s,t)-cut problem. Recall that an (s,t)-cut contains at least one edge from every (s,t)-path. A *fractional* (s,t)*-cut* is a fractional combination of edges $y : E \to \mathbb{R}_{\geq 0}$ that contains (in sum) one unit of edges from every (s,t)-paths. An edge cost $c(e)$ are now interpreted as the cost of one unit $y(e)$. All put together, the fractional relaxation of minimum cost (s,t)-cut is given by the following problem:

$$\text{minimize } \sum_{e \in E} c(e) y_e \text{ over } y : E \to \mathbb{R}_{\geq 0} \text{ s.t. } \sum_{e \in p} y_e \geq 1 \text{ for all } p \in \mathcal{P}_{s,t}. \tag{11.2}$$

(11.2) has m variables but exponentially many constraints. This setup leads to the following situation. We encourage the reader to pause and consider the following question herself before reading on.

> *Suppose you were given a vector $y \in \mathbb{R}^E_{\geq 0}$. How would you verify, in polynomial time, that y is a feasible solution? In particular, how does one verify that for every (s,t)-path p, the sum of y_e's over $e \in p$ is at least 1? (Is it even possible?)*

The question is nontrivial because there is not enough time to enumerate every (s,t)-path. But let us reformulate the question slightly: verifying every path p has $\sum_{e \in p} y_e \geq 1$ is the same as verifying that the *minimum* $\sum_{e \in p} y_e$, over all $p \in \mathcal{P}_{s,t}$, is at least 1. Let us reinterpret the values $y : E \to \mathbb{R}_{\geq 0}$ as *edge lengths*. Then the covering constraint is really saying that the *length of the shortest* (s,t)*-path w/r/t edge lengths* y_e*, is* ≥ 1; we can verify this constraint by computing the shortest (s,t)-path w/r/t y. An equivalent formulation of (11.2), then, is as follows.

> *Find the minimum cost set of edge lengths $y : E \to \mathbb{R}_{\geq 0}$ subject to s and t having distance 1 in the shortest path metric induced by y.*

This problem, besides being a relaxation of (s,t)-cut, is a very natural problem in its own right. For this reason, and to help distinguish the continuous nature of (11.2) for the discrete min-cut problem, we will also refer to the fractional min-cut problem as the (s,t)*-minimum cost metric problem*. We will write (Min-Metric) to denote both the optimization problem and the value of the optimization problem formulated in (11.2) above.

11.1.5 Linear programming and LP duality

The fractional versions of the (s,t)-path packing and cut problem described above are examples of *linear programs*, a class of mathematical optimization problems previously introduced in Chapter 6. We briefly review the basics.

Linear programs (LP's) are *constrained* continuous optimization problems where the goal is to (a) select a vector $x \in \mathbb{R}^n$ that (b) optimizes a linear objective subject to (c) linear equality and inequality constraints. That is, an optimization problem of the form

$$\begin{aligned} \text{min/max } \langle b, x\rangle = \sum_{j=1}^{n} b_j x_j \text{ over } x \in \mathbb{R}^n \\ \text{s.t. } A_1 x \leq c_1,\ A_2 x = c_2, \text{ and } A_3 x \geq c_3. \end{aligned}$$

where A_1, A_2, A_3 are matrices and b, c_1, c_2, c_3 are vectors.

Clearly, linear programs are useful for modeling real problems where we seek continuous solutions. Throughout this class we will encounter many different uses for LP's for understanding and solving *discrete* problems as well. A powerful feature of LP's is that they are *polynomial time solvable* [4], and conceptually it is easy to interact with these solvers as a black box. Moreover, real-world software for LP's is well-developed and reliable in practice.

We now introduce two canonical classes of LP's that capture most combinatorial problems.

Packing LPs. A *packing LP* is a linear program of the form

$$\max \langle b, x\rangle \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } Ax \leq c. \tag{P}$$

where $A \in \mathbb{R}^{m \times n}_{\geq 0}$, $b \in \mathbb{R}^n_{>0}$, and $c \in \mathbb{R}^m_{>0}$ all have nonnegative coefficients. We let $\textsc{Opt}(\text{P})$ denote the optimum value of the LP (P).

The fractional path packing problem is our first example of an packing LP. For path packing, we have one variable for every path. Identifying edges and paths as coordinates, then, we have:

1. $b = \mathbb{1}^{\mathcal{P}_{s,t}}$, the all-ones vector in $\mathbb{R}^{\mathcal{P}_{s,t}}$

2. $c \in \mathbb{R}^E_{\geq 0}$ is the edge capacities.

3. $A \in \{0,1\}^{E \times \mathcal{P}_{s,t}}$ is the incidence matrix defined by

   $$A_{e,p} = \begin{cases} 1 & \text{if } e \in p \\ 0 & \text{if } e \notin p, \end{cases}$$

   for each edge $e \in E$ and path $p \in \mathcal{P}_{s,t}$.

[4]More precisely, they are weakly polynomial time solvable, meaning the running times are polynomial in the bit complexity of the input.

Note that there are exponentially many variables in this LP so we could not even write it down in full in polynomial time, let alone apply a black box LP solver. Fortunately there are otherwise to solve the LP, as we will see.

Covering LPs. A *covering LP* is a linear program of the form

$$\min \langle c, y\rangle \text{ over } y \in \mathbb{R}^m_{\geq 0} \text{ s.t. } A^T y \geq b, \tag{C}$$

where[5] $A \in \mathbb{R}^{m\times n}_{\geq 0}$, $b \in \mathbb{R}^n_{>0}$, and $c \in \mathbb{R}^m_{>0}$. We let $\textsc{Opt}(\mathsf{C})$ denote the optimum value of the LP (C).

The minimum cost metric problem above is our first example of a covering LP. For minimum cost metric, we have one variable/column for each edge, and one row/constraint for each (s,t)-path.

1. $c \in \mathbb{R}^E_{\geq 0}$ is the edge costs.
2. $b = \mathbb{1}^{\mathcal{P}_{s,t}}$ is the all-ones vector in $\mathbb{R}^{\mathcal{P}_{s,t}}$.
3. $A^T \in \{0,1\}^{\mathcal{P}_{s,t}\times E}$ is the $\{0,1\}$-incidence matrix defined by

   $$A^T_{p.e} = A_{e,p} = \begin{cases} 1 & \text{if } e \in p \\ 0 & \text{if } e \notin p, \end{cases}$$

   for each edge $e \in E$ and path $p \in \mathcal{P}_{s,t}$.

Note that A, b, c are the same between our two examples.

LP duality. LP duality is about the relationship between the linear programs (P) and (C), particular when the matrices and vectors A, b, c are the same for both problems. In this case (P) and (C) are said to be *dual* to one another.

Suppose we have dual pair of (P) and (C); i.e., A, b, c refer to the same objects in either problem. Let $x \in \mathbb{R}^n_{\geq 0}$ be any feasible solution to (P) and let $y \in \mathbb{R}^n_{\geq 0}$ be any feasible solution to (C). We have

$$\langle b, x\rangle \overset{\text{(a)}}{\leq} \left\langle A^T y, x\right\rangle \overset{\text{(b)}}{=} \langle y, Ax\rangle \overset{\text{(c)}}{\leq} \langle y, c\rangle.$$

Here (a) is because $x \geq \mathbb{0}$ and $A^T y \geq b$. (b) is by definition of the transpose. (c) is because $y \geq \mathbb{0}$ and $Ax \leq c$. Thus, for a packing problem (P) and a covering problem (C) linked by duality, we have

$$\textsc{Opt}(\mathsf{P}) \leq \textsc{Opt}(\mathsf{C}).$$

[5]Of course, in (C), we could have written A instead of its transpose A^T, and swapped b and c, which would more closely resemble (P). It is convenient for the subsequent discussion on LP duality for A, b and c to have the same dimensions in (P) and (C).

If this argument seems familiar, it is because we just saw it for packing and covering paths in Section 11.1.2 above.

We ask the same question for packing and covering LP's as we did for packing and covering paths. When, if ever, is $\text{OPT}(\text{P}) = \text{OPT}(\text{C})$? The all-important *LP duality theorem* (here restricted to packing and covering problems) states that in fact they are always equal.

Theorem 11.2 (LP Duality for packing and covering)**.** $\text{OPT}(\text{P}) = \text{OPT}(\text{C})$.

We note that Theorem 11.2 holds even if A, b, and c have negative coefficients.

We will see that LP duality has important consequences for many combinatorial problems of interest – starting with max flow in the present discussion. Recall that (Max-Flow) is a packing LP and (Min-Metric) is a covering LP. Moreover, they are dual to one another. The LP duality theorem then tells us that (Max-Flow) = (Min-Metric). As a relaxation, we also have that (Min-Metric) $\leq$ (Min-Cut). That is:

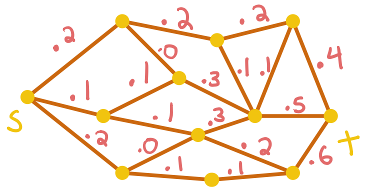
$$(\text{Max-Flow}) = (\text{Min-Metric}) \leq (\text{Min-Cut}).$$

11.1.6 Max-flow min-cut via LP duality

We now prove the following well-know *max-flow min-cut* theorem.

Theorem 11.3 (Menger [Men27] and Ford and Fulkerson [FF56])**.** (Max-Flow) = (Min-Cut).

Typically this theorem is proven algorithmically by the Ford-Fulkerson algorithm. Here we given an alternative proof based on LP duality[6]. Having already established that (Max-Flow) = (Min-Metric) $\leq$ (Min-Cut)[7], it suffices to prove that (Min-Cut) $\leq$ (Min-Metric).



Let $y \in \mathbb{R}^E_{\geq 0}$ be an optimum solution to the minimum cut LP, (11.2). We claim that

[6]For a video, see `https://youtu.be/J4yUdABv1tE`.
[7]With very little effort, thanks to LP duality

> *Given a fractional min-cut y, we can find a discrete (s,t)-cut $C \subseteq E$ with total capacity, $\sum_{e \in C} c(e)$, less than or equal to the fractional capacity of y, $\langle c, y \rangle$.*

This is our first example of *rounding* a fractional solution to a discrete one.

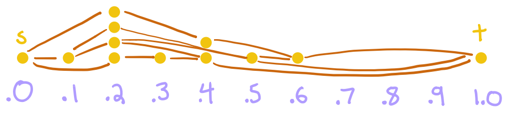
Notation. Before proceeding, we introduce some standard notation for cuts. For a set of vertices $S \subset V$, the *directed out-cut of* S, consisting of edges leaving S, is denoted by

$$\delta^+(S) \overset{\text{def}}{=} \{(u,v) \in E : u \in S,\, v \notin S\}$$

The *(directed) in-cut of* S, consisting of edges entering S, is denoted by

$$\delta^-(S) \overset{\text{def}}{=} \{(u,v) \in E : u \notin S,\, v \in S\}.$$

11.1.7 Line embeddings and sweep cuts.



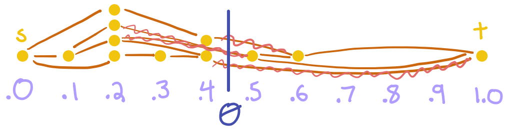
Recall that y also gives a set of edge lengths where the length of the shortest (s,t)-path is 1. We leverage this insight to *embed* the vertices V on the real line – assigning values $\alpha : V \to [0, +\infty)$ – as follows.

For each vertex v, let α_v be the length of the shortest $s \rightsquigarrow v$ path w/r/t the edge lengths $y \in \mathbb{R}^n_{\geq 0}$. We have $\alpha_s = 0$. We also have

$$\alpha_t = \min_{p \in \mathcal{P}_{s,t}} \sum_{e \in p} y_e \geq 1$$

because of the covering constraints in (Min-Metric).

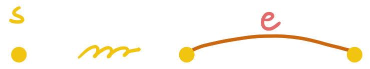
Consider the following *random cut*. We pick a value $\theta \in (0,1)$ uniformly at random. Let $S = \{v : x_v \leq \theta\}$, and let $\bar{S} = V \setminus S$. Since $\alpha_s = 0$ and $\alpha_t \geq 1$, the set of edges from S to $\bar{S}$ is always an (s,t)-cut.



The value of this cut is randomized. Let us bound the cost of the directed cut from S to $\bar{S}$, *in expectation*. We have

$$\mathbf{E}\left[\sum_{e\in\delta^+(S)} c(e)\right] \overset{\text{(a)}}{=} \sum_{e\in E} c(e)\,\mathbf{P}\Big[e\in\delta^+(S)\Big] \overset{\text{(b)}}{\leq} \sum_{e\in E} c(e) y_e = \text{(Min-Metric)} \tag{11.3}$$

Here (a) is by linearity of expectation. The key inequality, (b), is based on the shortest path metric.



For an edge $e = \{u,v\}$, we have $e \in \delta^+(S)$ iff $\alpha_u \leq \theta \leq \alpha_v$, which happens with probability at most $\alpha_v - \alpha_u$. Consider the shortest path from s to u w/r/t y, which has length α_u. Concatenating the shortest path from s to u with the edge e,

$$s \overset{\alpha_u}{\rightsquigarrow} u \xrightarrow{y_e} v,$$

gives a walk of length $\alpha_u + y_e$, hence $\alpha_v \leq \alpha_u + y_e$.

Consider now the inequality obtained in (11.3),

$$\mathbf{E}\left[\sum_{e\in\delta^+(S)} c(e)\right] \leq \text{(Min-Metric)}.$$

We have generated a *randomized (discrete) cut* that is *on average* no worse than the minimum fractional minimum cut. *By the probabilistic method, there exists a value θ where the (s,t)-cut has value at most this average.* If not, then the average would have to be higher. This establishes the existence of a minimum cut with cost equal to the (s,t)-minimum cost metric, and establishes the max-flow min-cut theorem. To extract the cut, one can simply scan θ over the interval $(0,1)$ and check all $n-1$ possible cuts. (In fact any $\theta \in (0,1)$ will work; see exercise C.28.)

11.2 Sparsest cut

Let G be undirected, and let $b : \binom{V}{2} \to \mathbb{R}_{\geq 0}$ be a set of nonnegative demands. Given a set S, the *sparsity of S* is defined as the ratio

$$\frac{c(\delta(S))}{\sum_{u\in S, v\notin S} b(u,v)}.$$

(For $S = \emptyset$ or $S = V$, we treat the sparsity of S as $+\infty$). The sparsest cut problem is to compute the set S of minimum sparsity. An important special case is *uniform sparsest cut* where the demands are uniformly $b(u, v) = 1$. Then the sparsity has the simpler form

$$\frac{c(\delta(S))}{|S||\bar{S}|}.$$

To minimize the uniform sparsity, we (of course) want to minimize the numerator and maximize the denominator. Minimizing the numerator is to find small cuts (as usual). The denominator (by AM-GM) is maximized by choosing $|S| \approx n/2$. So the uniform sparsest cut is lookingfor a tradeoff between the capacity of the cut and how "balanced" the cut is. (In fact, sparsest cut is used as a subroutine for the *balanced cut* problem, as we will discuss.)

To drive this point further, observe that

$$\frac{1}{n} \cdot \frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}} \leq \frac{c(\delta(S))}{|S||\bar{S}|} \leq \frac{2}{n} \cdot \frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}}$$

because $n/2 < \max\{|S|, |\bar{S}|\} \leq n$. That is, up to a constant factor, we are trying to minimize the ratio

$$\frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}}.$$

Here we clearly see the expense of choosing a very small set S.

In this chapter, we describe a very influential result of Leighton and Rao [LR99] that obtains a deterministic $O(\log(n))$ approximation ratio for the uniform sparsest cut problem. Their algorithm is based on applying region growing to the metric induced by the dual LP. Given our interest in randomized algorithms, we first present an alternative, randomized $O(\log(n))$ approximation algorithm for general demands based on *ℓ_1-metric embeddings*. We will also discuss lower bounds and some applications of sparsest cut.

11.2.1 The LP

Leighton and Rao's algorithm [LR99], as well as the randomized algorithm via metric embeddings, are both based on rounding an LP relaxation of the sparsest cut problem. However, since the sparsest cut optimizes a ratio, obtaining the linear relaxation is not as obvious. As a step in this general direction, consider the following (nonlinear) relaxation of the sparsest cut problem.

> *Compute a metric $d : V \times V \to \mathbb{R}_{\geq 0}$ minimizing the ratio*
> $$\frac{\sum_{e=\{u,v\}} c(e)d(u,v)}{\sum_{\{u,v\}} b(u,v)d(u,v)}.$$

Now, we can scale the distances up or down with no effect on the ratio. In particular, we can fix the denominator to be 1, which gives the following optimization problem which *is* a linear program.

> *Find the minimum c-cost metric such that the b-weighted sum of distances is at least* 1.

That is:

$$\begin{aligned} \text{minimize } & \sum_{e=\{u,v\}\in E} c(e)d(u,v) \\ \text{over all metrics } & d : V \times V \to \mathbb{R}_{\geq 0} \\ \text{s.t. } & \sum_{\{s,t\}} b(s,t)d(s,t) \geq 1. \end{aligned} \tag{11.4}$$

11.2.2 The dual LP and concurrent flow

To obtain the dual, it is helpful to rewrite (11.4) as a pure covering problem. Recall the correspondence between metrics and edge lengths, via shortest path distances. Then (11.4) is the same as:

> *Find the minimum cost edge lengths such that the b-weighted sum of shortest path distances is at least* 1.

To make this more explicit, for $s,t \in V$, let $\mathcal{P}_{s,t}$ denote the family of all (s,t)-paths. Let us define a *path bundle* as a collection of paths P consisting of an (s,t)-path $P_{s,t} \in \mathcal{P}_{s,t}$ for every pair (s,t). We let $\mathcal{P}_* \overset{\text{def}}{=} \prod_{\{s,t\}} \mathcal{P}_{s,t}$ denote the family of all path bundles. Then we can express the problem above as follows.

$$\begin{aligned} \text{minimize } & \sum_{e\in E} c(e)y(e) \text{ over } y : E \to \mathbb{R}_{\geq 0} \\ \text{s.t. } & \sum_{\{s,t\}} b(s,t) \sum_{e\in P_{s,t}} y(e) \geq 1 \text{ for all } P \in \mathcal{P}_*. \end{aligned} \tag{11.5}$$

We can separate this LP by computing the shortest (s,t)-path for every pair (s,t), and verifying the covering constraint for this bundle of shortest paths.

(11.5) covers path bundles with edges; thus, the dual packing LP packs path bundles into edges.

$$\begin{aligned} &\text{maximize} \sum_{P \in \mathcal{P}_*} x(P) \text{ over } x : \mathcal{P}_* \to \mathbb{R}_{\geq 0} \\ &\text{s.t.} \sum_{P \in \mathcal{P}_*} x(P) \sum_{\{s,t\} : e \in P_{s,t}} b(s,t) \leq c(e) \text{ for all } e \in E. \end{aligned} \tag{11.6}$$

In (11.6), each path bundle P represents a choice of paths for every (s,t)-pair to *concurrently* route the demands b. So (11.6) is trying to concurrently route b as much as possible subject to the capacity constraints. This problem is called *concurrent flow* or *demand multicommodity flow*.

11.3 Rounding via L_1-metric embeddings

We now analyze an approach to rounding the sparsest-cut metric based on *randomized embeddings*. This version readily generalizes to general demands $b : \binom{V}{2} \to \mathbb{R}_{\geq 0}$.

Let d be any metric that is a feasible solution to LP (11.4). Our goal is to convert d to a cut with sparsity comparable to the cost of d.

11.3.1 Rounding line embeddings

We first observe that some special cases of metrics are very easy to round – namely, those related to line embeddings. Suppose there is a function $f : V \to \mathbb{R}$ such that

$$d(u,v) = |f(u) - f(v)| \qquad \text{for all } u, v \in V.$$

(Such a function f, placing V on the real line, is called a *line embedding*.) Rescaling and translating (which does not effect the sparsity), we may assume that $\min_u f(u) = 0$, and $\max_v f(v) = 1$.

Consider the following random cut S (which we have seen before in proving the max-flow min-cut theorem). Pick $\theta \in (0,1)$ uniformly at random, and let $S = \{u : f(u) \leq \theta\}$. Observe that for each edge $e = \{u,v\}$, we have

$$\mathbf{P}[e \in \delta(S)] = |f(u) - f(v)| = d(u,v).$$

Thus we can rewrite the sparsity of d as

$$(\text{sparsity of } d) = \frac{\sum_{\{u,v\} \in E} c(u,v) d(u,v)}{\sum_{\{s,t\}} d(s,t) b(s,t)} = \frac{\mathbf{E}\left[\sum_{e \in \delta(S)} c(e)\right]}{\mathbf{E}\left[\sum_{s \in S, t \in \bar{S}} b(s,t)\right]}. \tag{11.7}$$

Note that the RHS is *not* the expected sparsity of S. The expected sparsity of S is the quantity

$$\mathbf{E}[\text{sparsity of } S] = \mathbf{E}\left[\frac{\sum_{e \in \delta(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s,t)}\right],$$

which is not the same as the quantities in (11.7). This is in contrast to our proof of max-flow min-cut, where S is a minimum (s,t)-cut on average, and the existence of a minimum (s,t)-cut follows immediately from the probabilistic method.

Still the probabilistic approach can be salvaged with a little more work. Observe that S can only be one of $n-1$ different sets $S_1, \dots, S_{n-1}$ where $\emptyset \subsetneq S_1 \subset S_2 \subset S_3 \cdots \subset S_{n-1} \subsetneq V$. For each i, let $p_i = \mathbf{P}[S = S_i]$. Then

$$(11.7) = \frac{\sum_{i=1}^{n-1} p_i \sum_{e \in \delta(S_i)} c(e)}{\sum_{i=1}^{n-1} p_i \sum_{s \in S_i, t \in \bar{S}_i} b(s,t)}.$$

Now we apply the following elementary fact. (The proof is left as exercise C.69.)

Lemma 11.4. *Let $a_1, \dots, a_h, b_1, \dots, b_h > 0$. Then*

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i} \leq \max_i \frac{a_i}{b_i}.$$

It follows that for some S_i, the sparsity of S_i is at most the sparsity of d. In conclusion, we have shown the following.

Lemma 11.5. *Let $d : \binom{V}{2} \to \mathbb{R}$ be a metric induced by a line embedding. Then one can partition the vertices into two sets $(S, \bar{S})$ such that*

$$\frac{\sum_{e \in \delta(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s,t)} \leq \frac{\sum_{\{u,v\} \in E} c(u,v) d(u,v)}{\sum_{\{s,t\}} d(u,v) b(s,t)}.$$

Rounding L_1-metrics So much for metrics given by line embeddings. How about a metric obtained as a sum of line metrics? Recall that the L_1-metric on $\mathbb{R}^h$ is defined by

$$\|x - y\|_1 = \sum_{i=1}^{h} |x_i - y_i|.$$

Suppose d was the L_1-metric of an embedding $f : V \to \mathbb{R}^h$.That is,

$$d(u,v) = \|f(u) - f(v)\|_1 = \sum_{i=1}^{h} |f_i(u) - f_i(v)|$$

for a function $f : V \to \mathbb{R}^h$. We can think of this as the sum of h line metrics $f_1, \dots, f_h$. The sparsity of d expands out to

$$\text{(sparsity of } d) = \frac{\sum_{i=1}^h \sum_{e=\{u,v\}} c(e)|f_i(u) - f_i(v)|}{\sum_{i=1}^h \sum_{\{u,v\}} b(u,v)|f_i(u) - f_i(v)|}.$$

Applying Lemma 11.4 again, we see that one of these line embeddings, say f_j, has sparsity no worse then d. From the line embedding $f_j : V \to \mathbb{R}$, we can extract a cut with sparsity at most that of f_j. This establishes the following.

Lemma 11.6. *Let $d : \binom{V}{2} \to \mathbb{R}$ be the L_1-metric over an explicit embedding of V. Then one can partition the vertices into two sets $(S, \bar{S})$ such that*

$$\frac{\sum_{e\in\delta(S)} c(e)}{\sum_{s\in S, t\in\bar{S}} b(s,t)} \leq \frac{\sum_{\{u,v\}\in E} c(u,v)d(u,v)}{\sum_{\{s,t\}} d(u,v)b(s,t)}.$$

To sum up: L_1 metrics can be rounded without loss. We can find an L_1-metric with sparsity within a factor α of the sparsest metric d, then we can convert that into an α-approximate sparsest cut.

11.3.2 Randomized L_1-metric embeddings

We now know that L_1-metrics can be rounded to sparse cuts without any loss. But the LP for sparsest cut produces a generic metric d, that is not an L_1-metric.

Our new strategy, given a generic metric d, is to try to find an L_1-metric d_1 with sparsity comparable to d. We then invoke Lemma 11.6 to round obtain a sparse cut from d_1. Our L_1-metric d_1 will be defined by a mapping $f : V \to \mathbb{R}^h$ (for some $h \in \mathbb{N}$), so that

$$d_1(u,v) = \|f(u) - f(v)\|_1.$$

We will prove the following theorem.

Theorem 11.7. *Let $d : \binom{V}{2} \to \mathbb{R}_{\geq 0}$ be a metric and $\delta \in (0,1)$. For $h = O(\log(n)\log(1/\delta))$, one can construct a randomized embedding $f : V \to \mathbb{R}^h$ such that for all $u, v \in V$, we have*

$$\|f(u) - f(v)\|_1 \leq O(\log(1/\delta))d(u,v)$$

deterministically, and

$$\mathbf{P}[\|f(u) - f(v)\|_1 \leq d(u,v)] \leq \delta \tag{11.8}$$

For $\delta = 1/\operatorname{poly}(n)$ and $h = O\left(\log^2 n\right)$, we can apply the union bound to (11.8) over all pairs u, v. This gives the following theorem.

Corollary 11.8. *Let $d : \binom{V}{2} \to \mathbb{R}_{\geq 0}$ be a metric and $\delta \in (0,1)$. For $h = O\left(\log^2(n)\right)$, one can construct a randomized embedding $f : V \to \mathbb{R}^h$ such that with high probability, for all $u, v \in V$,*

$$d(u,v) \leq \|f(u) - f(v)\|_1 \leq O(\log n) d(u,v).$$

The embedding $f : V \to \mathbb{R}^h$ described in Corollary 11.8 is said to be a $O(\log n)$-*distortion metric embedding* as it maps points in one metric space into another while preserving all distances up to a $O(\log n)$-multiplicative factor.

The $O(\log n)$-distortion embedding into L_1 is the last ingredient for the following algorithm for sparsest cut.

1. Solve the LP (11.4) to obtain a sparsest metric d.
2. Invoke Corollary 11.8 to obtain a $O(\log n)$-distortion embedding $f : \binom{V}{2} \to \mathbb{R}^h$ from d into the L_1-metric. The L_1-metric via f has sparsity at most a $O(\log n)$ factor greater than d.
3. Invoke Lemma 11.6 to round the L_1-metric to a cut with sparsity at most the metric, which is a factor $O(\log n)$ greater than the sparsity of d (and the optimum of (11.4)).

This algorithm is due to Linial, London, and Rabinovich [LLR95], and establishes the following.

Theorem 11.9. *There is a $O(\log n)$ randomized approximation to (non-uniform) sparsest cut (on undirected graphs).*

Actually, Linial, London, and Rabinovich [LLR95] observed that one can do slightly better when there are demands for only k commodity pairs. (i.e., k pairs (u,v) with $b(u,v) > 0$.)

Theorem 11.10. *There is a $O(\log k)$ randomized approximation to sparsest cut, where k is the number of commodity pairs with nonzero demand.*

This approximation factor is obtained by building on the ideas in Theorem 11.9, and left as exercise C.21.

Now we describe the randomized algorithm of Linial, London, and Rabinovich [LLR95] that computes the embedding in Theorem 11.14. We note that previously Bourgain [Bou85] had obtained a deterministic embedding but the output dimension

```
random-Fréchet($d : V \times V \to \mathbb{R}_{\geq 0}$)
1. for $i = 1, \dots, \lceil \log n \rceil$
   A. let $S_i$ sample each $v \in V$ independently with probability $2^{-i}$
   B. for each $v \in V$
      1. $f_i(v) \leftarrow \min_{s \in S_i} d(s, v)$
2. return $f : V \to \mathbb{R}_{\geq 0}^{\lceil \log n \rceil}$
```

Figure 11.1: A $O(\log n)$ dimension, randomized Frechét embedding with $O(\log n)$ distortion in expectation

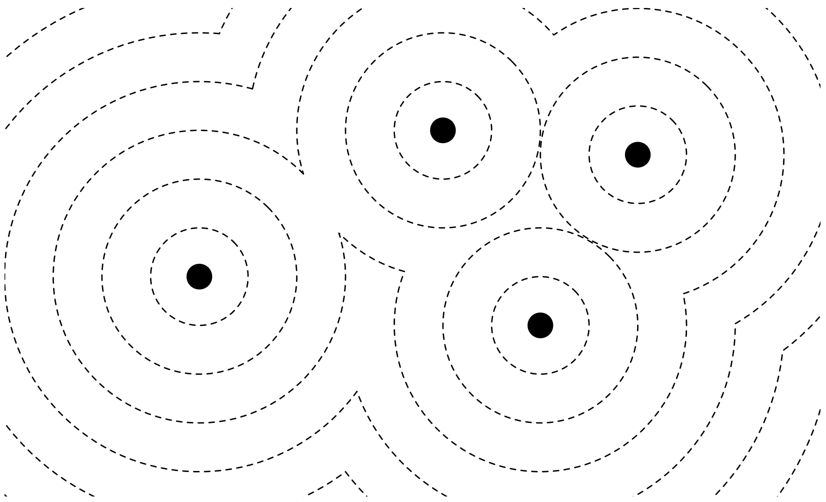


Figure 11.2: Level sets by distance from a set of points S, encoding one coordinate of a Frechét embedding.

h was exponential. Linial, London, and Rabinovich's algorithm [LLR95] can be interpreted as an efficient randomized implementation of Bourgain's embedding [Bou85].

The algorithm, which we call `random-Fréchet`, is extremely simple. We generate $\lceil \log n \rceil$ coordinates. For $i = 1, \dots, n$, we sample a set S_i where each point is sampled independently with probability $1/2^i$. For each vertex v, we find the distance between v and (the closest point in) S_i. This gives the ith coordinate of v. Pseudocode is described in Fig. 11.1.

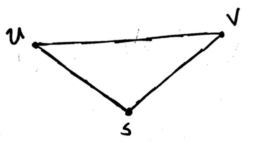
Each coordinate of the randomized embedding is given by distances from a set. There is a name for this class of embeddings: *Fréchet embeddings*. Fig. 11.2 attempts to visualize a single coordinate generated in this manner.

11.3.3 Low-distortion in expectation

We now turn to proving Theorem 11.14. Consider an instance of the `random-Frechet` algorithm, which computes a randomized embedding $f : V \to \mathbb{R}_{\geq 0}^{O(\log(n))}$.

For ease of notation, for a vertex v and coordinate i, we let $v_i \overset{\text{def}}{=} f_i(v)$ denote the ith coordinate of the embedding of v.

Lemma 11.11. *For $u, v \in V$ and $i \in \mathbb{N}$, $|u_i - v_i| \leq d(u, v)$.*



Proof. By the triangle inequality, we have both

$$d(s,u) - d(s,v) \leq d(u,v) \text{ and } d(s,v) - d(s,u) \leq d(u,v).$$

for all $s \in S_i$. □

Lemma 11.11 implies that $\|u - v\|_1 \leq O(\log n) d(u, v)$, since there are $O(\log n)$ dimensions and each contributes at most $d(u, v)$. The lower bound is harder: informally, we want to show $\|u - v\|_1 \geq d(u, v)$, up to constant factors. This lower bound is too strong; instead, we settle for the same inequality but only in expectation.

Lemma 11.12. *For $u, v \in V$, we have $\mathbf{E}[\|u - v\|_1] \geq c d(u, v)$ for some constant $c > 0$.*

Proof. For ease of notation, let $\delta = d(u, v)$. For each i, let r_i be the minimum length r such that there are at least 2^i points at distance $\leq r$ from u, and 2^i points at distance $\leq r$ from v; i.e.,

$$r_i = \arg\min_{r>0} \Big\{ |\{x : d(u,x) \leq r\}| \geq 2^i, |\{x : d(v,x) \leq r\}| \geq 2^i \Big\}$$

We claim that

> *For each index i, we have*
>
> $$|u_{i+1} - v_{i+1}| \geq (\min\{r_i, \delta/2\} - \min\{r_{i-1}, \delta/2\})$$
>
> *with constant probability $c > 0$.*

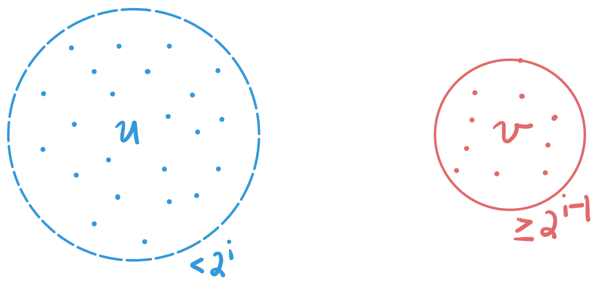
Before proving the claim, suppose it holds true. Let k be the largest index such that $r_{k-1} \leq \delta/2$ (for which the claim applies). We have

$$\begin{aligned}
\mathbf{E}[\|u-v\|_1] &\geq \sum_{i=0}^{k} \mathbf{E}[|u_{i+1} - v_{i+1}|] \\
&\overset{\text{(a)}}{\geq} c \sum_{i=0}^{k} (\min\{r_i, \delta/2\} - \min\{r_{i-1}, \delta/2\}) \\
&\overset{\text{(b)}}{\geq} \frac{c\delta}{4},
\end{aligned}$$

as desired. Here (a) applies the claim and (b) is by telescoping sums and recalling that $r_{k+1} > \delta/2$.

It remains to prove the claim. We have two cases: *(a)* $r_i \leq \delta/2$, and *(b)* $r_{i-1} < \delta/2 \leq r_i$. We assume without loss of generality that r_i is defined by u; i.e., $|\{x : d(u,c) < r_i\}| < 2^i$.

Case 1: $r_i \leq \delta/2$. Let $U = \{x : d(u,x) < r_i\}$, and let $V = \{x : d(v,x) \leq r_{i-1}\}$. We have $|U| < 2^i$ and $|V| \geq 2^{i-1}$. Since $r_{i-1} < r_i \leq \delta/2$, U and V are disjoint.

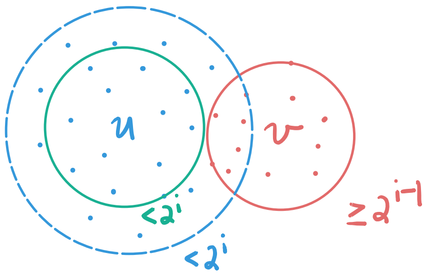


S_{i+1} samples each point with probability 2^{-i-1}. By direct calculation, S_{i+1} samples no points from U with constant probability, and at least one point from V with constant probability. Since U and V are disjoint, whether any point from U is sampled and whether any point from V is sampled is independent. Thus S_{i+1} samples a point from V and no points from U simultaneously with some constant probability $c > 0$. In this event, we have $u_{i+1} \geq r_i$ and $v_{i+1} \leq r_{i-1}$, so $u_{i+1} - v_{i+1} \geq r_i - r_{i-1}$. In expectation, we have

$$\mathbf{E}[|u_{i+1} - v_{i+1}|] \geq c(r_i - r_{i-1}),$$

as desired.

Case 2: $r_i > \delta/2 > r_{i-1}$. Let $U = \{x : d(u,x) \leq \delta/2\}$ (with $\delta/2$ in place of r_i) and let $V = \{x : d(u,v) \leq r_{i-1}/2\}$.



By the same argument as above, we have that S_{i+1} samples a point from V and no points in U with some constant probability $c > 0$. In this event, $u_{i+1} - v_{i+1} \geq \delta/2 - r_{i-1}$. ☐

Theorem 11.13. `randomized-Frechet` *embeds V into $\mathbb{R}^{\lceil \log n \rceil}$ such that*

$$\|u - v\|_1 \leq \lceil \log n \rceil d(u,v) \text{ and } \mathbf{E}[\|u - v\|_1] \geq c d(u,v) \text{ for all } u, v \in V,$$

for some absolute constant $c > 0$.

11.3.4 Amplification

Theorem 11.13 shows that a single instance of `randomized-Fréchet` obtains $O(\log n)$ distortion "in expectation", so to speak, for each pair of vertices. In particular the embedded distance is bounded above deterministically but below only in expectation. We want to strengthen this so that the lower bound holds with high probability.

Theorem 11.14. *With probability of error $1/\operatorname{poly}(n)$, the average of $O(\log n)$ embeddings produced by* `randomized-Frechet` *is an embedding V into $\mathbb{R}^{O(\log^2 n)}$ such that*

$$c d(u,v) \leq \|u - v\|_1 \leq C \log(n) d(u,v) \text{ for all } u, v \in V$$

for absolute constants $c, C > 0$.

Proof sketch. Fix $u, v \in V$. We treat each coordinate difference $|u_i - v_i|$ (for $O(\log^2 n)$ coordinates over $O(\log n)$ independent calls to `random-Frechet`) as an independent random variable bounded above by $d(u,v)$. The expected sum of the $|u_i - v_i|$'s is $\Omega(n \log(n))$. By standard Chernoff inequalities, the sum is strongly concentrated at the mean; scaling down by $\log n$ (from averaging) gives the desired result. ☐

11.4 Application: Minimum bisection

A *bisection* is a partition of the vertices V into $(S, \bar{S})$ of (essentially) equal size: $\lfloor n/2 \rfloor \leq |S|, |\bar{S}| \leq \lceil n/2 \rceil$. Alternatively a bisection can be defined in terms of cuts as a set of edges whose removal leaves the graph with connected components of at most $\lceil n/2 \rceil$ vertices each. In any case the minimum bisection problem is to compute a vertex set $S \subset V$ of size $|S| = \lfloor n/2 \rfloor$ minimizing the cost of the cut, $c(\delta(S))$.

There is a natural connection between minimum bisection and the sparsest cut – the minimum bisection problem can be recast as the restricting the sparsest cut problem to vertex sets with exactly half the vertices.

The following algorithm uses a $O(\log n)$-approximation for uniform sparsest to obtain a *bicriteria*-approximation algorithm. In particular, it returns a (1/3)-balanced partition $(S, \bar{S})$ – that is, $n/3 \leq |S| \leq 2n/3$ – with cost at most $O(\log n)$ times the cost of the minimum bisection. The algorithm is very simple. It repeatedly computes the sparsest cut and removes the smaller side from the graph, until the number of vertices removed is at least $n/3$ (and necessarily at most $2n/3$).

```
1. For $i = 1, 2, \ldots$
   A. $S_i \leftarrow$ smaller side of a $O(\log n)$-approximate uniform sparsest cut.
   B. If $|S_1| \cup \cdots \cup |S_i| \geq n/3$
      1. return $(S_1 \cup \cdots \cup S_i, V - (S_1 \cup \cdots \cup S_i))$.
   C. Else remove $S_i$ and all incident edges form the graph, and repeat.
```

It is (relatively) easy to see why the algorithm returns a (1/3)-balanced cut; it remains to show that the cost is comparable to that of the minimum bisection. The intuition is as follows. Suppose for simplicity we have an exact algorithm for the sparsest cut. The sparsest cut is very close to the minimum (weighted) expansion, which we recall is the cost of the cut divided by the number of vertices on the smaller side of the cut. In particular this ratio for the sparsest cut is no worse than that of the minimum bisection. If the sparsest cut is balanced, then its cost is comparable to the minimum cost bisection. While its not balanced, we can interpret the sparsest cut as removing some vertices from the graph at the cost of the edges being cut. The ratio of vertices removed per unit cost – the bang-for-buck, so to speak – is at least as good. So we are gradually removing vertices while paying a favorable rate compared to the minimum cost bisection.

There are some additional details to take care of – for one, the minimum bisection in the input graph may no longer be a minimum bisection in the residual graphs, although it will still be somewhat balanced as long as we haven't removed $n/3$ vertices yet. Also we only have a $O(\log n)$-approximation for the sparsest cut, which will imply that we pay an additional $O(\log n)$ factor throughout the argument. Exercise C.29 guides the reader through a formal proof of the argument.

Theorem 11.15. *In polynomial time, one can compute* $(1/3)$*-balanced cut with total cost at most a* $O(\log n)$*-factor greater than the minimum bisection.*

11.5 Additional notes and materials

One can do better than a $O(\log n)$ approximation for uniform sparsest cut – Arora, Rao, and Vazirani [ARV09] gave a $O\left(\sqrt{\log n}\right)$ via semi-definite programming and ideas from high-dimensional geometry. We may discuss this result later in the course; in the meantime we refer the reader to lecture notes by Rothvoss [Rot16].

Lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

11.6 Exercises

Exercise 11.1. Recall the randomized rounding based proof of the max-flow min-cut theorem. Recall that we analyzed a random cut which was based on a threshold $\theta \in (0,1)$ chosen uniformly at random. Prove or disprove that for (essentially) *all* $\theta \in (0,1)$, the corresponding cut is a minimum $\{s,t\}$-cut.

Exercise 11.2. In most textbooks, max flow is presented as the following LP, which in particular has polynomial size in the input graph G.

$$\begin{aligned} \text{maximize } & \sum_{e \in \delta^+(s)} z_e - \sum_{e \in \delta^-(s)} z_e \text{ over } z : E \to \mathbb{R}_{\geq 0} \\ \text{s.t. } & z_e \leq c(e) \text{ for all } e \in E \\ & \sum_{e \in \delta^+(v)} z_e = \sum_{e \in \delta^-(v)} z_e \text{ for all } v \in V \setminus \{s, t\}. \end{aligned} \tag{11.9}$$

The second set of constraints are called *flow conservation* constraints. Show that the above LP is equivalent to the (fractional path packing version of) (Max-Flow) in the following sense.

1. Show that for every (feasible) fractional path packing P, there is a feasible solution z to (C.2) with the same objective value.
2. Show that for every feasible solution z to (C.2), there is a feasible path packing P with the same objective value.[8]

Exercise 11.3. Prove Lemma 11.4.

Exercise 11.4. Recall the randomized $O(\log n)$ approximation algorithm for sparsest cut based on L_1-embeddings. Note that this is also logarithmic in the number of demand pairs, $\binom{n}{2}$. We consider the case where the demands are *sparse*; more precisely, where there are at most k commodities (i.e., pairs) $\{s, t\}$ with nonzero demand $b(s, t) > 0$.

1. Prove the following extension of Theorem 11.14:

   Let $d : V \times V \to \mathbb{R}_{\geq 0}$, and let $T \subseteq V$ be a subset of points with $\ell = |T|$. Then for $h = O\left(\log^2 \ell\right)$, one can compute a randomized embedding $f : V \to \mathbb{R}^h$ such that:

   (a) For all $u, v \in V$, $\|f(u) - f(v)\|_1 \leq O(\log \ell) d(u, v)$ (always).
   (b) With high probability, for all $s, t \in T$, $\|f(s) - f(t)\|_1 \geq d(s, t)$.

2. Describe (rigorously) how to adjust the algorithm and analysis from Section 11.3 to obtain a randomized $O(\log k)$ approximation factor, where k is the number of commodities with nonzero demand.

[8]The second problem is trickier than the first. One should be able to prove it using only the ideas and results in this chapter (without retracing the flow algorithms of ensuing chapters).

Exercise 11.5. Recall the bicriteria approximation algorithm for the minimum bisection problem from Section 11.4.

1. Show that the algorithm returns a 1/3-balanced cut.
2. For each iteration i, w/r/t the graph remaining at iteration i, we have
$$\frac{c(\delta(S_i))}{|S_i|} \leq O(\log(n))\frac{\textsc{Opt}}{n}.$$
3. Combine the two parts above to prove that the algorithm returns a 1/3-balanced cut of size $O(\log n)\textsc{Opt}$.

Exercise 11.6. One can also consider the bisection problem in directed graphs. Here the goal is to find a vertex set S of size $\lfloor n/2 \rfloor \leq |S| \leq \lceil n/2 \rceil$ minimizing the cost of the directed cut $c(\delta^+(S))$. Suppose one had access to a $O(\log n)$ approximation algorithm for uniform directed sparsest cut (as described in **??**). Using this as a subroutine, design and analyze an algorithm that obtains a bicriteria approximation algorithm for the minimum directed bisection problem with essentially the same approximation bicriteria for the undirected setting: compute a set S with $n/3 \leq |S| \leq 2n/3$ with cost $c(\delta^+(S))$ at most a $O(\log n)$-factor greater than that of the minimum directed bisection.

Chapter 12

Tree Metrics

12.1 Introduction

Many flow and cut problems are ultimately about paths, whether packing paths in flow, cutting paths in cuts, evaluating shortest paths in fractional cuts, etc. Obviously there are many paths between any two points in a graph and this makes all these problems nontrivial. An extremely simple setting, then, would be a graph where there is a unique path between any two vertices: by definition, a *tree*. Many graphs problems become trivial in a tree. Here we will study a bold approach to graph algorithms based on this idea: process the input graph G to produce a tree T that preserves its salient properties, solve the problem on T, and lift the solution back to G. Of course a tree T cannot preserve all of G, so we will only preserve specific properties and only approximately at that, in such a way that is appropriate to the problem at hand.

In this discussion, we will focus on preserving the shortest path metric of a graph. Let G be an undirected graph, and let d_G denote the shortest path metric in G. Let T be a spanning tree of T (with the same edge weights), and let d_T denote the shortest path metric in T. (Of course, the shortest path in T is also the only path.) Ideally, we want $d_T(u, v)$ to resemble $d_G(u, v)$ as much as possible for each edge $e = \{u, v\}$. In general, we have

$$d_G(u, v) \leq d_T(u, v) \text{ for all } u, v \in V$$

simply because T is a subgraph of G. For an edge $e = \{u, v\}$, the *stretch* of e is defined as the ratio

$$(\text{stretch } e) \overset{\text{def}}{=} \frac{d_T(u, v)}{d_G(u, v)}.$$

We say that d_T has *uniform stretch* (at most) α, for $\alpha \geq 1$, if every edge has stretch at most α.

A natural goal is to obtain a spanning tree with small uniform stretch. However the n-vertex cycle C_n presents a lower bound of $n-1$. Indeed, any spanning tree T of C_n is obtained by dropping one edge e; this edge e is stretched around the cycle, so to speak, and has stretch $n-1$.

The n-vertex cycle C_n indicates that we cannot, in the worst-case, find spanning trees with uniform stretch better than $n-1$. (We leave it as exercise C.34 to obtain a matching upper bound.) The rest of this chapter discusses two different approaches that obtain better bounds for relaxations of this problem.

Low-stretch spanning trees. Alon, Karp, Peleg, and West [AKP+95] showed how to compute a spanning tree T where the average stretch among all edges is $n^{o(1)}$, in the sense that

$$\frac{1}{|E|}\sum_{e\in E}(\text{stretch } e) \leq n^{o(1)}$$

Dominating tree metrics. Bartal [Bar96; Bar98] ignored the requirement that T is a spanning tree of G; more generally he sought auxiliary trees T where the vertices of G correspond to the leaves of T, while retaining the property T that $d_T \geq d_G$. He produced *randomized* trees where *for each edge* the average stretch was polylog(n):

$$\mathop{\mathbf{E}}_{T}[(\text{stretch } e)] \leq O(\text{polylog}(n)) \text{ for all } e \in E.$$

Note that this a different sense of "average stretch" then above. We will present an algorithm of [FRT04] building on [Bar98] to obtain (per-edge) average stretch of $O(\log n)$.

12.2 Low-Stretch Spanning Trees

We first present the low-stretch spanning trees of [AKP+95]. Here we recall that we want to compute a spanning tree with low stretch on average over all the edges.

Suppose our goal was to obtain average stretch (roughly) D for a parameter $D > 0$. Let us partition the graph in vertex-disjoint subgraphs each with radius at most $D/2$ from some center vertex, and compute a shortest path tree from each center. Then every edge *within* a neighborhood has stretch at most D; it remains to address the edges that are cut by the partition. We can try to address these edges recursively by contracting every subgraph/subtree into a single vertex, leaving a multigraph G' consisting of the cut edges, and recursing on this graph. This produces a tree T' on the contracted multigraph G'; expanding out the vertices of T' by the underlying shortest path trees gives a spanning tree T. Recursively, we might expect that every

[AKP+95] algorithm for $m^{o(1)}$-average stretch spanning trees:

1. *Compute a low diameter decomposition:* repeatedly, until no vertices remain:
   - A. Select a remaining vertex v and compute a shortest path tree of v in the remaining graph.
   - B. Remove the set C_v all vertices (remaining) in V within some distance $D' \leq D$ such that

     $$|\delta(C_v)| \leq O(\log(m))(1 + |E[C_v]|)$$

     where $E[C_v]$ denotes the set of all (remaining) edges incident to some vertex in v.
2. *Recurse:* Let G' be the multi-graph obtained from the input graph by contracting each C_v to a single vertex. Recurse on G' to obtain a spanning tree T', and return the tree T obtained by replacing each C_v with the corresponding shortest path tree.

---

edge cut edge e has stretch $O(D)$ in T', but this expands out to stretch $O(D^2)$ with respect to T because passing through a vertex in T' actually corresponds to traversing a path of length $O(D)$ in the underlying tree.

So we have a problem where each step of the recursion induces an additional factor of D. Now, recall that we want to preserve *average* stretch. Let $f(m)$ be the stretch obtained by the recursive approach. We have

$$f(m) \leq D(\#\text{ internal edges}) + (D+1)f(\#\text{ external edges}).$$

Studying this recursion, we can see that if the number of external edges was extremely small, then we might hope that it can offset the extra factor of D. How can we minimize the number of exteneral, or cut, edges? Region growing! Low diameter decompositions!

If we partition the graph by region growing techniques, we can ensure that

$$(\#\text{ external edges}) \leq \frac{O(\log m)}{D} m.$$

This revises the recursive bound as

$$f(m) \leq Dm + (D+1)f(c \log(m) m/D)$$

for a constant $c > 0$. For $D = e^{\sqrt{\log(m)/\log\log m}}$ the recursion is bounded by $f(m) = e^{O\left(\sqrt{\log m \log\log m}\right)}$. (The calculations are given below.)

Theorem 12.1. *The AKPW algorithm returns a spanning tree with average stretch* $e^{O\left(\sqrt{\log m \log\log m}\right)}$.

Solving the recurrence. We have

$$f(m) \leq Dm + (D+1)f(c_0 \log(m)m/D)$$

where $\epsilon = c_0 \log(m)/D$ for a constant c_0. The height of the recursion is

$$h = O\Big(\log_{D/c_0\log(m)} m\Big) = O(\log(m)/\log(D/c_0\log(m))) = O\left(\frac{\log(m)}{\log(D)}\right)$$

assuming $D = \Omega(\log m)$. Unrolling the recursion gives

$$f(m) \leq O(Dm)\sum_{i=0}^{h}((1+1/D)c_0\log(m))^i \leq Dme^{O(h\log\log m)}.$$

To minimize the RHS, we can instead minimize the logarithm of the RHS, $\log(m) + \log(D) + O(h\log\log m)$. Choosing D to make the last two terms (roughly) equal, we have

$$\log(D) = \frac{\log(m)\log\log(m)}{\log(D)},$$

hence

$$\log(D) = \sqrt{\log(m)\log\log(m)}.$$

Then $D = e^{\sqrt{\log(m)\log\log(m)}}$ gives

$$f(m) \leq me^{O\left(\sqrt{\log(m)\log\log(m)}\right)},$$

as desired.

12.3 Hierarchical Tree Metrics

The [AKP+95] algorithm was able to obtain low stretch *in total*. This means that a few unlucky edges might have extremely high stretch. In this section we want every edge to have low stretch, in some sense. Unfortunately we already know that it is impossible to guarantee $o(n)$ stretch for every edge simultaneously. But a different possibility opens up when we allow for randomization. Perhaps we can output a *randomized* tree T, such that for each edge e, the expected stretch of e is $o(n)$. Such a claim does *not* contradict the lower bound for the n-vertex cycle; in fact, one can get constant expected stretch for the cycle which we leave as exercise C.67.

To build some intuition, let us start from the [AKP+95] algorithm for inspiration (even though ultimately we will not produce a spanning tree). A high level goal is to inject randomization so that every edge has a decent chance at having low stretch. To this end there are at least two natural ways to introduce randomization into [AKP+95].

1. We can make the radii of the clusters randomized, instead of a deterministic function of the total number of edges cut.

2. Second, the order of vertices that center the clusters can be randomized, which would seem most equitable.

Both of these ideas will be reflected at a high level in the following algorithm which we now present.

The algorithm we present will produce a randomized *hierarchical tree metric* over V. This means that the tree T will be rooted, with V at the leaves, and the edges between height i and height $i-1$ have length α^i for a fixed constant α. Here we choose $\alpha = 4$ to simplify calculations, though we note that $\alpha = 2$ is more common, and the analysis can be adjusted to accommodate any fixed constant. The convenience of a hierarchical tree T is that the tree distance $d_T(u,v)$ is entirely determined by the height of their least common ancestor, and within a constant factor of the biggest edges at the top of the corresponding subtree. This additional structure turns out to be useful for several other problems.

We now present [FRT04]'s randomized algorithm. We assume the input is an edge-weighted graph where the minimum edge length is normalized to 1. We let $D = \max_{u,v} d(u,v)$ denote the diameter of the graph. Below we describe the algorithm in detail and first we give a high level description. For every radius of the form $\alpha 4^i$, we are randomly scooping out balls of size $\alpha 4^i$, where $\alpha \in [1,2]$ is drawn uniformly at random, and the vertices at the center of the balls are in random order. A key and subtle point is that we use the *same* (random) α and ordering for every i. the intersections of these balls (across i's) induce a laminar family of sets over V which are arranged as a tree.

[FRT04]'s algorithm producing a randomized hierarchical tree metric.

1. Let $v_1, \ldots, v_n$ be a uniformly random ordering of V. Let $L = \lceil \log_4 D \rceil$. Let $\alpha \in [1, 2]$ be drawn uniformly at random.
2. For i from L down to 0,
   (a) For each vertex v_j in order,
      i. Let $C_{i,j}$ be the set of vertices at distance at most $\alpha 4^{i-1}$ from v_j, excluding any vertex already included by the cluster of a previous $C_{i,j}$.
3. We use the $C_{i,j}$'s to arrange the vertices as leaves in a tree T hierarchically as follows. For each intermediate node x at height i, the leaves in the subtree rooted at x corresponds to a set of vertices with diameter at most 4^i. The root at height $L + 1$ corresponds to V. The nodes at height L correspond to the clusters $C_{L,j}$. In general, for a node x at height i corresponding to a set $S \subseteq V$, its children correspond to the (nonempty) intersections of S with clusters $C_{\ell,j}$. (Here a cluster center v_j may not be in S.) Observe the leaves (at height 0) each correspond to a single vertex V because $C_{0,j} = \{v_j\}$ for all j. Each edge descending from height i is given weight 4^i.

So much for the algorithm. Here, then, is the key claim.

Theorem 12.2. *[FRT04]'s algorithm produces a randomized hierarchical tree T such that for each edge e, $\mathbf{E}[(\text{stretch } e)] \leq O(\log n)$.*

To prove the theorem, fix an edge $e = \{u, v\}$. Recall that $d_T(u, v)$ is decided, up to a constant factor, by the height k where u and v are first separated. (Then $d(u, v) = O\left(4^k\right)$.) When this occurs, u and v are separated in particular by a cluster of radius $\alpha 4^k$ centered at some vertex w; in this event, we say that w "contributes" 4^k to $d(u, v)$ (which upper bounds the diameter of the remaining vertices). By this terminology, we have

$$d_T(u, v) \leq \sum_w O(1)(\text{contrib. of } w \text{ to } d_T(u, v)). \tag{12.1}$$

Now, fix a vertex w. Suppose that w was the ℓth closest vertex to u or v (i.e., with respect to $\min\{d(w,u), d(w,v)\}$). The key lemma, which we analyze below, is that

$$\mathbf{E}[\text{contrib. from } w \text{ to } d_T(u,v)] \leq O(1/\ell) d(u,v).$$

Taking expectations of Eq. (12.1) and applying the bound above to each w gives $O(\log n)$ stretch, as desired.

It remains to prove the key lemma, as follows.

Lemma 12.3. *Let w be the ℓth closest vertex to u or v. Then*

$$\mathbf{E}[\textit{contrib. from } w \textit{ to } d_T(u,v)] \leq 4d(u,v)/\ell.$$

Proof. We assume without loss of generality that w is closer to u than v (i.e., $d(u,w) \leq d(v,w)$). We first observe that w contibutes to $d_T(u,v)$ only if the following two events both occur.

- E_1: $d(w,u) \leq \alpha 4^k \leq d(w,v)$ for some k.
- E_2: w is ordered before any of the $\ell - 1$ vertices that are closer to u or v.

Indeed, the necessity of the first condition is clear. The second is necessary because any closer vertex would otherwise cluster either u or v (or both) before w.

We also observe that the above conditions are *independent*, since the first event depends (only) on α and the second event depends on the random ordering, which are independent. It is also clear that the second event E_2 occurs with probability $1/\ell$. It remains to analyze E_1. We have two cases.

Case 1: $d(u,v) \geq 4d(w,u)$. Then for any k satisfying the inequality in E_1, this inequality and the triangle inequality imply that

$$4^{k+1} \leq 2d(w,v) \leq 2(d(w,u) + d(u,v)) \leq (5/2)d(u,v).$$

Thus the contribution from w is at most $O(d(u,v))$. It follows that

$$\mathbf{E}[\text{contrib. from } w \text{ to } d_T(u,v)] \leq 2.5d(u,v)\,\mathbf{P}[E_2] = 2.5d(u,v)/\ell,$$

as desired.

Case 2: $d(u,v) \leq d(w,u)$. We first note that there may not be any k in Item E_1 satisfying inequality E_1; if not, then the claim is immediate. Henceforth we assume such a k exists. We claim that the choice of k is unique. Indeed, suppose there exists some choice of k and $\alpha \in [1,2]$ such that the inequality in E_1 holds. We have

$$d(w,u) \overset{(a)}{\geq} d(w,v) - d(u,v) \overset{(b)}{>} 4^k - 4^k/2 > 2 \cdot 4^{k-1}$$

which rules out smaller values of k. Here (a) is by the the triangle inequality and (b) is by assumption on α. To rule out larger values of k, we have

$$d(w,v) \leq d(w,u) + d(u,v) < 4^{k+1}$$

by similar reasoning.

Thus the choice of k in E_1 is unique; fix k as such. Now we have

$$\mathbf{P}[E_1] \leq \frac{|[d(w,u), d(w,v)]|}{|[4^k, 2 \cdot 4^k]|} \overset{(c)}{\leq} \frac{d(u,v)}{4^k}$$

by (c) the triangle inequality. Thus

$$\mathbf{E}[\text{contrib. from } w \ \ldots] \leq 4^{k+1}\, \mathbf{P}[E_1]\, \mathbf{P}[E_2] \leq 4d(u,v)/\ell,$$

as desired. □

Remark 12.4. When we apply this algorithm on the shortest path metric of a graph $G = (V,E)$, we may also want, for every pair of vertices u, v, a walk $w_T : u \rightsquigarrow v$ in G of length at most $d_T(u,v)$. To this end, when computing T, observe that each "cluster center" corresponds to a vertex v, and when we have cluster with center u and a child cluster with center v, the underlying metric implies a path from u to v with length at most the length assigned to the auxiliary parent edge between the clusters. Thus, given two vertices u and v, we can take the unique path between the leaves u and v in T, and concatenate the underlying walks in G to get a walk from u to v in G with length at most $d_T(u,v)$.

12.4 Additional notes and materials

Lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

12.5 Exercises

Exercise 12.1. Design and analyze an algorithm that computes a spanning tree with uniform stretch $n-1$ (matching the lower bound induced by the cycle).

Exercise 12.2. For the n-vertex cycle C_n, describe a randomized tree metric where each edge has expected stretch $O(1)$.

Exercise 12.3. Prove that the $O(\log n)$ bound is tight for tree metrics (up to constants).

Exercise 12.4. Recall that the low-stretch spanning tree of [AKP+95] obtained average stretch $n^{o(1)}$. We consider extensions to the weighted average. Let $w(e) \in \mathbb{R}_{>0}$ be a positive weight for every edge, and let $W = \sum_{e \in E} w(e)$ be the total weight. We assume for simplicity that the weights are between 1 and poly(n). We still treat the edges as unit length edges. Design and analyze and algorithm to compute a spanning tree T such that

$$\frac{1}{W} \sum_{e \in E} w(e)(\text{stretch } e) \leq n^{o(1)}.$$

Exercise 12.5. Show how to use the randomized tree metric to randomly round the sparsest cut LP and obtain a $O(\log n)$-approximation for sparsest cut.[1]

Exercise 12.6. The following problem, called the *buy-at-bulk network design* problem, models the situation where you are building out a network that supports communication between various terminal pairs with minimum total cost, with "economies of scale" built into the cost function.

Formally, we have an undirected graph $G = (V, E)$ with edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$. There are k terminal pairs $(s_1, t_1), \dots, (s_k, t_k)$, each associated with a demand $d_i > 0$. The high-level goal is to choose a walk $w_i : s_i \rightsquigarrow t_i$ for all i minimizing the "cost" of

[1]Try to prove tree metrics are also L_1-metrics.

buying enough capacity of each capacity to route all these paths simultaneously. To model the cost, we are given a monotone subadditive function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, and for each edge, we pay $f(u)$ for capacity u per unit length. *Montone* means that $f(u)$ is nondecreasing in u. *Subadditive* means that $f(u_1 + u_2) \leq f(u_1) + f(u_2)$ for $u, v \geq 0$.

Now, each edge must have enough capacity to route all the demand of all the paths using it. Thus the total cost of a selection of paths $\{p_i : s_i \rightsquigarrow t_i : i = 1, \dots, k\}$ is

$$\sum_{e \in E} \ell(e) f\left(\sum_{i : e \in w_i} d_i\right).$$

This problem is NP-Hard, so instead we will try to approximate it. Our goal is to design and analyze a polynomial time approximation algorithm for the buy-at-bulk network design problem.

We will use the randomized tree metric of Section 12.3. The algorithm is as follows. We first compute a randomized tree metric d_T, with underlying weighted tree $(T = (V_T, E_T), \ell_T : E_T \to \mathbb{R}_{\geq 0})$, over the shortest path metric of G with respect to ℓ. Treat T as a graph with edge lengths ℓ_T, and mapping the terminal pairs (s_i, t_i) to the corresponding leaves of T, we solve the buy-at-bulk network design problem over T with the same cost function f. We then map this solution back to walks in G and return this solution.

To elaborate on the second point, recall that every edge node in T is associated with a cluster centered at some vertex in G. Each edge e in T is supported by the shortest path in G between the vertices associated with the endpoints of e. Denoting this path by P_e for $e \in E_T$, we have $\ell(P_e) \leq \ell_T(e)$. In this way, every path P in T maps to a walk W in G obtained by concatenating the walks supporting the (tree) edges in P.

We break down the analysis into the following steps.

1. Design and analyze an algorithm that solves the buy-at-bulk network design problem exactly in a tree. (This shows that the first step is exact.)

2. Describe a mapping from solutions in G to solutions in the randomized tree metric T such that for a fixed solution in G, in expectation (over T), the cost of the mapped solution in T is at most a $O(\log n)$ factor greater than original cost in G.

3. Show that for every solution in T, there is a solution in G where the cost in G is at most the cost in T.

4. Combine the three parts above to prove that the randomized solution returned by our algorithm has expected cost $O(\log n)$ OPT.

Chapter 13

Sampling geometric range spaces

13.1 Introduction

Many problems in computational geometry take place in the context of a *range space*. A range space (P, R) consists of a collection of *points* P and a family of *ranges* R, which are subsets of points. (P and R need not be distinct or finite.) For example, P may be a collection of n points in $\mathbb{R}^2$, and R may be the family of all closed discs in $\mathbb{R}^2$. Typical queries, for a given $r \in R$, include:

1. Is r empty?
2. How many points does r contain?
3. Does r contain at least an ϵ-fraction of P, for given ϵ?

Extensions of this model include weighted points and ranges, or a distribution over points instead of a finite set. We focus on the unweighted and discrete setting for simplicity.

For a fixed range space (P, R) with P finite, we define the *measure* $\mu(r)$ of a range $r \in R$ as the fraction of all the points it contains:

$$\mu(r) = \frac{|r \cap P|}{|P|}.$$

One can interpret $\mu(r)$ as the probability that a random point from P likes in r.

This chapter is about small random samples from P that still approximately preserving the measure of every range $r \in R$, for a broad and geometrically natural class of "low complexity" range spaces (to be defined later). To formalize this, for a set of points Q, let μ_Q denote the measure with respect to Q:

$$\mu_Q(r) = \frac{|r \cap Q|}{|Q|}.$$

We say that Q is an *ϵ-sample* for (P, R) if it approximates the measure up to an ϵ-additive error; that is, rb

$$|\mu(r) - \mu_Q(r)| \leq \epsilon \text{ for all } r \in R.$$

Q is an *ϵ-net* if it hits all ranges with measure at least ϵ:

$$\mu_Q(r) > 0 \text{ for all } r \in R \text{ with } \mu(r) > \epsilon.$$

Note that an ϵ-sample is stronger than an ϵ-net. ϵ-nets are useful in situations where you want to identify "heavy-hitter" range spaces.

Computing a small ϵ-sample is impossible with no restriction on R. For example, suppose $R = 2^P$ is the family of all subsets of P. Then for any proper subset $Q \subseteq P$ with $|Q| < (1-\epsilon)|P|$, Q cannot be an ϵ-sample because in particular it will fail for the range $r = P \setminus Q$. For the same reason, any ϵ-net Q of P must have at least $(1-\epsilon)|P|$ points.

For a subset of points $Q \subseteq P$, let

$$R \wedge Q = \{r \cap Q : r \in R\}$$

denote the family of subsets of Q induced by R. For many natural range spaces, especially in low-dimensional geometry, $R \wedge Q$ is a very small subset of 2^Q. Above we mentioned the setting of disks in the plane. The intersection of a disk with the point set Q gives a subset of Q, but in general, disks cannot induce all possible subsets of a set of points. In fact, consider 4 points arranged in a square. It is impossible to take a disk and overlay it so that it only covers two opposite corners of the square. Thus disks are inherently of limited complexity, and we hope to leverage this when sampling.

One way to model this complexity is via the *growth function*. The growth function $g : \mathbb{N} \to \mathbb{N}$ models the maximum cardinality of $R \wedge Q$ as a function of $|Q|$. For $k \in \mathbb{N}$, $g(k)$ is defined by

$$g(k) \overset{\text{def}}{=} \max\{|R \wedge Q| : Q \subseteq P, |Q| \leq k\}.$$

(P, R) is said to have *polynomial growth of degree d* if

$$g(k) \leq O\left(k^d\right).$$

For example, disks in the plane have polynomial growth of degree 3 (as we prove in Section 13.3). The family of closed halfspaces in $\mathbb{R}^d$ have polynomial growth of degree

$d+1$. There are several ways to establish whether a range space has polynomial growth, via parameters such as the *VC-dimension* (Section 13.3) and the *shattering dimension* (see [Har18b]).

Now, let (P,R) be a range space with growth function $g(n)$. Let $\epsilon, \delta \in (0,1)$, and let $Q \subset P$ be a random sample of P. We want to understand how big Q should be so that it is either an ϵ-sample or an ϵ-net with probability at least $1-\delta$.

A straightforward approach to this question is to apply the following *additive Chernoff bound.*

Theorem 13.1. *Let $X_1, \dots, X_n \in [0,1]$ be independent, $\mu = \mathbf{E}[X_1 + \cdots + X_n]/n$ and $\epsilon > 0$. Then*

$$\left.\begin{array}{l}\mathbf{P}\left[\frac{1}{n}(X_1 + \cdots + X_n) \geq \mu + \epsilon\right] \\ \mathbf{P}\left[\frac{1}{n}(X_1 + \cdots + X_n) \leq \mu - \epsilon\right]\end{array}\right\} \leq e^{-2\epsilon^2 n}.$$

The proof, which is similar to that of the multiplicative Chernoff bound, is deferred to Section 13.A.

Suppose we want an ϵ-sample for the range space (R,P) with growth function $g(n)$. Let $Q \subset P$ be a random sample (with repetition) of k points and let μ_Q be the corresponding measure. Fix a range r and consider $\mu_Q(r)$. We have $\mathbf{E}[\mu_Q(r)] = \mu(r)$. Write

$$\mu_Q(r) = \frac{1}{k}(X_1 + \cdots + X_k)$$

where $X_i = 1$ if the ith sampled point lies in r, and 0 otherwise. Thus

$$\mathbf{P}[|\mu_Q(r) - \mu(r)| \geq \epsilon] = \mathbf{P}\left[\left|\frac{1}{k}(X_1 + \cdots + X_k) - \mu(r)\right| \geq \epsilon\right] \leq 2e^{-\epsilon^2 k}.$$

Now, there are effectively at most $|R \wedge P| \leq g(|P|)$ distinct ranges we want to preserve. For a given parameter $\delta \in (0,1)$, and $k = \log(g(|P|)/2\delta)/\epsilon^2$ we have

$$\mathbf{P}[Q \text{ is } \textit{not} \text{ an } \epsilon\text{-sample}] \overset{\text{(a)}}{\leq} \sum_{r \in R \wedge P} \mathbf{P}[|\mu_Q(r) - \mu(r)| \geq \epsilon] \leq |R \wedge P| \cdot \frac{\delta}{g(|P|)} \leq \delta$$

by (a) the union bound. That is, $\log(g(|P|)/2\delta)/\epsilon^2$ points suffice to give an ϵ-sample with probability at least $1-\delta$. For example, for disks in the plane, we need to sample $O(\log(|P|)/2\epsilon^2)$ points to obtain an ϵ-sample with constant probability.

Meanwhile, for ϵ-nets, one can show a random sample of size $O(\log(g(|P|))/\epsilon)$ is an ϵ-net with constant probability in a relatively straightforward fashion. (See exercise C.50.)

It turns out that one can do much better than these initial bounds, and surprisingly, remove the dependence on $|P|$ entirely. These stronger sampling bounds, called the *ϵ-sample theorem* and *ϵ-net*, are the main topic of this chapter. For ϵ-samples we have the following [VC71].

Theorem 13.2. *Let (P, R) be a range space with growth function $g(n)$, and let $\epsilon, \delta \in (0,1)$ be given. Let $\ell \in \mathbb{N}$ such that*

$$\ell \geq C \log(g(2\ell)/\delta)/\epsilon^2$$

for a universal constant C. Then a random sample of ℓ points with repetition from P is an ϵ-sample with probability at least $1 - \delta$.

For polynomial growth we have:

Corollary 13.3. *Let (P, R) be a range space with polynomial growth function $g(n) = O\left(n^d\right)$, for fixed d, and let $\epsilon, \delta \in (0,1)$ be given. Let $\ell \in \mathbb{N}$ such that*

$$\ell \geq \frac{C}{\epsilon^2}(d \log(d/\epsilon) + \log(1/\delta)),$$

for a universal constant C. Then a random sample of ℓ points with repetition from P is an ϵ-sample with probability at least $1 - \delta$.

We prove Theorem 13.2 in Section 13.2. Note that $|Q|$ is independent of $|P|$. For ϵ-nets an even smaller sample suffices [HW87]:

Theorem 13.4. *Let (P, R) be a range space with growth function $g(n)$, and let $\epsilon, \delta \in (0,1)$ be given. Let $\ell \in \mathbb{N}$ such that*

$$\ell \geq C \log(g(2\ell)/\delta)/\epsilon$$

for a universal constant C. Then a random sample of ℓ points with repetition from P is an ϵ-net with probability at least $1 - \delta$.

For polynomial growth we have:

Corollary 13.5. *Let (P, R) be a range space with polynomial growth function $g(n) = O\left(n^d\right)$, for fixed d, and let $\epsilon, \delta \in (0,1)$ be given. Let $\ell \in \mathbb{N}$ such that*

$$\ell \geq \frac{C}{\epsilon}(d \log(d/\epsilon) + \log(1/\delta)),$$

for a universal constant C. Then a random sample of ℓ points with repetition from P is an ϵ-net with probability at least $1 - \delta$.

The proof of Theorem 13.4 is similar to the proof of Theorem 13.2, and left to the reader in exercise C.53.

We mention that the theorems above can be formulated in a continuous setting, where instead of a finite point set P, we have a distribution D of points. Here the measure $\mu(r)$ of a range r is the probability of a point drawn from D lying in r. The proofs of the ϵ-sample and ϵ-net theorems presented here extend immediately to these settings. We focus on the discrete setting as it captures the essential ideas while being simpler to discuss.

13.2 Proof of the ϵ-sample theorem

In the section we prove the ϵ-sample theorem. We first restate the theorem for the reader's convenience.

Theorem 13.2. *Let (P, R) be a range space with growth function $g(n)$, and let $\epsilon, \delta \in (0, 1)$ be given. Let $\ell \in \mathbb{N}$ such that*

$$\ell \geq C \log(g(2\ell)/\delta)/\epsilon^2$$

for a universal constant C. Then a random sample of ℓ points with repetition from P is an ϵ-sample with probability at least $1 - \delta$.

Proof. Let A be the event that Q_1 is incorrect for some $r \in R$.

$$A \overset{\text{def}}{=} \text{the event that } |\mu_1(r) - \mu(r)| > \epsilon \text{ for some } r.$$

We want to show that $\mathbf{P}[A] < \delta$.

Let Q_2 be a second, independent random sample of the same size. Define B by

$$B \overset{\text{def}}{=} \text{the event that } |\mu_2(r) - \mu_1(r)| > \epsilon/2 \text{ for some } r \in R.$$

Define $C = A \wedge B$ as the event where the conditions of events A and B hold simultaneously for the same $r \in R$; i.e., $|\mu_1(r) - \mu(r)| > \epsilon$ *and* $|\mu_1(r) - \mu_2(r)| > \epsilon/2$ for some $r \in R$.

We claim that $\mathbf{P}[A] \leq 2\,\mathbf{P}[B]$. To this end, we first write

$$\mathbf{P}[B] \geq \mathbf{P}[C] = \mathbf{P}[A, C] = \mathbf{P}[C \mid A]\,\mathbf{P}[A]. \tag{13.1}$$

Now, in event A, a range r is incorrectly measured by Q_1 by more than ϵ. That particular range r is correctly measured by Q_2 to within $\epsilon/2$ with probability at least $1/2$ by the additive Chernoff inequality. In this case we have

$$|\mu_1(r) - \mu_2(r)| \geq |\mu(r) - \mu_1(r)| - |\mu(r) - \mu_2(r)| > \epsilon - \epsilon/2 = \epsilon/2,$$

hence event C. Thus $\mathbf{P}[C \mid A] \geq 1/2$, which gives $\mathbf{P}[B] \geq \mathbf{P}[A]/2$ when plugged into (13.1).

Thus an upper bound on $\mathbf{P}[B]$ gives an upper bound on $\mathbf{P}[A]$ up to a factor of 2. To upper bound $\mathbf{P}[B]$, suppose we generate Q_1 and Q_2 alternatively as follows.

1. Sample $2k$ points Q_0 from P.
2. Randomly partition Q_0 in half. Let Q_1 be one half and let Q_2 be the other.

Since

$$\mathbf{P}[B] = \sum_{Q_0} \mathbf{P}[Q_0] \, \mathbf{P}[B \mid Q_0] \leq \max_{Q_0} \mathbf{P}[B \mid Q_0],$$

it suffices to upper bound the probability of B conditional on Q_0.

Fix Q_0. Observe that the restricted range space (Q_0, R) has at most $m \overset{\text{def}}{=} g(2\ell)$ distinct ranges over Q_0 (!).

Let μ_0 be the measure with respect to Q_0. For each r, we have

$$\mathbf{P}[|\mu_1(r) - \mu_0(r)| \geq \epsilon/4] \leq \frac{\delta}{4m}$$

as well as

$$\mathbf{P}[|\mu_2(r) - \mu_0(r)| \geq \epsilon/4] \leq \frac{\delta}{4m}$$

by the additive Chernoff inequality. By taking the union bound over all m distinct ranges over Q_0 we have $|\mu_1(r) - \mu_0(r)| \leq \epsilon/4$ and $|\mu_2(r) - \mu_0(r)| \leq \epsilon/4$ for all $r \in R$ with combined probability of error at most $\delta/2$. In this case we have

$$|\mu_1(r) - \mu_2(r)| \leq |\mu_1(r) - \mu_0(r)| + |\mu_2(r) - \mu_0(r)| = \epsilon/2$$

for all $r \in R$; i.e., event B does not occur. Thus $\mathbf{P}[B] \leq \delta/2$, completing the proof. □

13.3 VC dimension

We have shown that very few points are needed for ϵ-samples and ϵ-nets when the range system (P, R) has small (e.g., polynomial) growth $g(n)$. But how do we bound the growth function? One way to do this is via a property called the *VC dimension* of (P, R).

For $Q \subset P$, we say that R *shatters* Q if the family of projections $R \wedge Q$ induces all $2^{|Q|}$ subsets of Q.

The *VC dimension* of (P, R) is the maximum cardinality $|Q|$ of any shattered subset $Q \subset P$.

Intervals. Let P be the real line $\mathbb{R}$, and R the family of intervals on $\mathbb{R}$. Any two points can be shattered by intervals, but given any three points $a < b < c$, it is impossible to induce the set $\{a, b\}$ with an interval.

Disks Let P be the plane $\mathbb{R}^2$, and let R be the family of closed disks. Any three points can be shattered by disks. Consider a set Q of four points. We have two cases.

In one case, one of the four points is contained in the convex hull of the others. Then it is impossible to find a disk that covers the three outer points without including the point in their convex hull, since disks are convex. (In general, any set of points that are not in convex position cannot be shattered by range spaces of convex shapes.)

Now suppose all four points are in convex position. Call these points $\{a, b, c, d\}$ in order along the convex hull. If we had two disks D_1 and D_2 that induced the "opposite" pairs $\{a, c\}$ and $\{b, d\}$, respectively, then the boundaries of D_1 and D_2 would intersect at four points. But two circles can only intersect at two points.

Halfspaces. Let $P = \mathbb{R}^d$, and let R be the set of closed halfspaces:

$$R = \left\{ \left\{ x \in \mathbb{R}^d : \langle a, x \rangle \leq b \right\} : a \in \mathbb{R}^d, b \in \mathbb{R} \right\}.$$

Here we have the following fact.

Theorem 13.6 (Radon's theorem)**.** *Let Q be a set of $d + 2$ points in $\mathbb{R}^d$. Then one can partition Q into two sets $S_1 \cup S_2$, such that* $\mathrm{conv}(S_1) \cap \mathrm{conv}(S_2) \neq \emptyset$*.*

Radon's theorem implies that the VC-dimension is at most $d + 1$. Indeed. if Q had $d + 2$ points, then partition $Q = S_1 \cup S_2$ as in Radon's theorem. If a halfspace H contains S_1, then it also contains a point in the convex hull of S_2. But then it H must also contain at least one point from S_2. Thus it is impossible to have $H \cap Q = S_1$. To show that VC-dimension is exactly $d + 1$, one verifies that the regular simplex in $\mathbb{R}^d$ with $d + 1$ vertices can be shattered by half spaces.

13.4 Sauer's lemma

Sauer's lemma connects VC-dimension to the sample techniques from the beginning of our discussion. It states that range spaces of fixed VC-dimension have polynomial growth, hence samples proportional to the VC-dimension (and independent of the number of points!) give ϵ-samples and ϵ-nets.

Theorem 13.7. *Let (P, R) be a range space of VC-dimension d_{VC}. Then (P, R) has growth function*

$$g(n) \leq \sum_{i=0}^{d_{\text{VC}}} \binom{n}{i} \leq n^{d_{\text{VC}}}.$$

Proof. Define $f(d_{\text{VC}}, n) = \sum_{i=0}^{d_{\text{VC}}} \binom{n}{i}$. We prove that $g(n) \leq f(d_{\text{VC}}, n)$ for all d_{VC} and all n by induction on n and d_{VC}. Clearly $g(n) \leq f(d_{\text{VC}}, n)$ when $n = 0$ or $d_{\text{VC}} = 0$.

Let $Q \subset P$ have n points and fix a point $p \in Q$. Without loss of generality we may assume that each range is a subset of Q, that is,

$$r \subseteq Q \text{ for all } r \in R.$$

(Otherwise replace R with $R \wedge Q$.)

Now, the high level idea is to reduce to range spaces over $Q - p$, and apply induction to each. Let

$$S \overset{\text{def}}{=} R \wedge (Q - p) = \{r - p : r \in R\}$$

i.e, we project R onto $Q - p$ we take each range r and by removing p from each range. Of course $|S| \leq |R|$. $|S|$ may be smaller because two ranges may project $r_1, r_2 \in R$ to the same range in $Q - p$; i.e., $r_1 - p = r_2 - p$. This occurs iff $r_1 = r_2 + p$ or $r_2 = r_1 + p$.

We make a second set of ranges T over $Q - p$ that consists of the projections that had two preimages in the projection; i.e.,

$$T = \{s \in S : s \in R,\ s + p \in R\}.$$

We have $|R| = |S| + |T|$.

Now, S has VC-dimension at most d because removing a point cannot increase the VC-dimension. Consider T. If T shatters a point set $X \subseteq Q - p$, then R shatters $X + p$, since for every $r \in T$, both $r \in R$ and $r + p \in R$. It follows that T has VC-dimension $d - 1$.

By induction, we have

$$|R| = |T| + |S| \leq f(d_{\text{VC}} - 1, n - 1) + f(d_{\text{VC}}, n - 1).$$

Moreover,

$$\begin{aligned} f(d_{\text{VC}} - 1, n - 1) + f(d_{\text{VC}}, n - 1) &= \sum_{i=0}^{d_{\text{VC}}-1} \binom{n-1}{i} + \sum_{i=0}^{d_{\text{VC}}} \binom{n-1}{i} \\ &= \sum_{i=1}^{d_{\text{VC}}} \binom{n-1}{i-1} + \sum_{i=0}^{d_{\text{VC}}} \binom{n-1}{i} \\ &\overset{\text{(a)}}{\leq} \sum_{i=0}^{d_{\text{VC}}} \binom{n}{i} = f(d_{\text{VC}}, n), \end{aligned}$$

as desired. To see (a), observe that the RHS counts the number of subsets of size i from a universe of n points. The LHS counts the same collection by counting the number of such sets that contains a given point in the first sum, and counting the number of such sets that omits a given point in the second sum. □

13.5 Additional notes and materials

See [Har18b; Har11] for additional background on geometric sampling and complexity, including another notion of geometric complexity called the *shattering dimension* which is omitted here. [Har18b; Har11] also describes an interesting connection between geometric sampling and *discrepancy*.

Lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

13.6 Exercises

Exercise 13.1. Provide directly (and without leveraging the techniques from Section 13.2) that sampling $O(\log(g(|P|))/\epsilon)$ points gives an ϵ-net with constant probability.

Exercise 13.2. This exercise asks you to prove the ϵ-net theorem, Theorem 13.4. One can prove it similarly to the ϵ-sample theorem, Theorem 13.2. Below we define the events A and B for you to get you started, and ask you to complete the proof (in full detail).

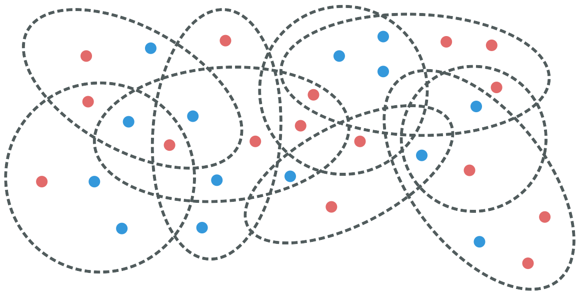
> Let Q_1 and Q_2 be two random samples of points (of appropriate size, TBD by you) inducing measures μ_1 and μ_2, respectively. Define the events A and B by:
>
> $A \overset{\text{def}}{=}$ the event that $Q_1 \cap r = \emptyset$ and $\mu(r) \geq \epsilon$ for some $r \in R$.
>
> $B \overset{\text{def}}{=}$ the event that $Q_1 \cap r = \emptyset$ and $\mu_2(r) > \epsilon/2$ for some $r \in R$.

Exercise 13.3. Recall that we can obtain a $O(\log m)$-approximation for set cover by randomly rounding the LP. There are also natural special cases where the points and sets have geometric structure; here we consider one such setting.

Suppose we have a set of m disks $D = \{d_1, \ldots, d_m \subseteq \mathbb{R}^2\}$, and n points $P = \{p_1, \ldots, p_n \in \mathbb{R}^2\}$. We say that a point p_j *hits* a disk D_i if $p_j \in D_i$. (You can assume you can query if p_j hits D_i in $O(1)$ time.) A set of points $Q \subseteq P$ is a *hitting set* of D if every disk is hit by some point in Q.

Consider the problem of computing the minimum cardinality hitting set Q of D. This is a special case of set cover, where points "cover" the disks that they hit. Design and analyze a (polynomial-time) approximation algorithm for this problem. Here, the smaller the approximation ratio, the better, but we will not emphasize constant factors. (Yes, you can do better than $O(\log m)$.)

Exercise 13.4. A classical case of *discrepancy* is the following "balanced covering problem". Suppose you have n sets over n points. The goal is to color all the points either red or blue, so that each set has the same number of points of each color, or as close to the same as possible.



Numerically this is generalized and modeled as follows. Let $A \in [0,1]^{n \times n}$ be a square matrix with bounded entries. For a vector of signs $x \in \{-1,+1\}^n$, the *discrepancy* of x is the quantity

$$\|Ax\|_\infty = \max_i |Ax|_i.$$

(If A is the $\{0,1\}$ incidence matrix of the n sets (as rows) to the n points (as columns), if we let $+1$ denote the color blue -1 denote the color red, then $\|Ax\|_\infty$ is the maximum color imbalance over all the sets.) The general goal is to chose $x \in \{-1,+1\}^n$ to minimize the discrepancy.

Here the goal is to establish a baseline for this problem. Design and analyze an algorithm that computes a point $x \in \{-1,1\}^n$ with discrepancy

$$\|Ax\|_\infty \leq O\left(\sqrt{n \log n}\right).$$

Later in the semester we will improve the bound fto $\|Ax\|_\infty \leq O(\sqrt{n})$.

13.A Proof of the additive Chernoff bound

In this section we prove the additive Chernoff bound. First we required the following upper bound on the moment generating function of a bounded random variable with mean 0.

Lemma 13.8. *Let $X \in [-1,1]$ be a bounded random variable with mean $\mathbf{E}[X] = 0$. Then for all $t \in \mathbb{R}$,*

$$\mathbf{E}\left[e^{tX}\right] \leq e^{t^2/2}.$$

Proof. Observe that for all $x \in [-1,1]$, we have

$$x = \frac{1+x}{2} - \frac{1-x}{2},$$

hence by convexity of $f(x) = e^{tx}$, we have

$$e^{tx} \leq \left(\frac{1-x}{2}\right)e^{-t} + \left(\frac{1+x}{2}\right)e^{t}.$$

Consequently

$$\mathbf{E}\left[e^{tX}\right] \leq \mathbf{E}\left[\left(\frac{1-X}{2}\right)e^{-t} + \left(\frac{1+X}{2}\right)e^{t}\right] = \frac{e^t + e^{-t}}{2}.$$

Moreover, by the Taylor expansion of e^x, we have

$$\begin{aligned}
\frac{1}{2}\left(e^t + e^{-t}\right) &= \frac{1}{2}\left(\left(1 + t + \frac{t^2}{2} + \frac{t^3}{3!} + \cdots\right) + \left(1 - 1 + \frac{t^2}{2} - \frac{t^3}{3!} + \cdots\right)\right) \\
&= 1 + \frac{t^2}{2!} + \frac{t^4}{4!} + \frac{t^6}{6!} + \cdots \\
&\leq 1 + \frac{t^2}{2} + \frac{(t^2/2)^2}{2!} + \frac{(t^2/2)^3}{3!} + \cdots \\
&= e^{t^2/2},
\end{aligned}$$

as desired. □

We now prove the additive Chernoff bound, which we first restate for the reader's convenience.

Theorem 13.1. *Let $X_1, \dots, X_n \in [0,1]$ be independent, $\mu = \mathbf{E}[X_1 + \cdots + X_n]/n$ and $\epsilon > 0$. Then*

$$\left.\begin{array}{l} \mathbf{P}\Big[\frac{1}{n}(X_1 + \cdots + X_n) \geq \mu + \epsilon\Big] \\ \mathbf{P}\Big[\frac{1}{n}(X_1 + \cdots + X_n) \leq \mu - \epsilon\Big] \end{array}\right\} \leq e^{-2\epsilon^2 n}.$$

Proof. We only prove the first bound; the second bound follows from the first by considering the variables $Y_i = 1 - X_i$. We also prove the inequality for a weaker constant of $1/2$ in the exponent instead of 2. Obtaining the factor of 2 requires more calculations and we refer the reader to [Har18a] for this proof.

The high-level proof idea is similar to the multiplicative proof idea – we exponentiate and apply Markov's inequality, and leverage independence to reduce the analysis to analyzing the moment generating function of each X_i separately.

For each X_i, let $\mu_i = \mathbf{E}[X_i]$. Thus $n\mu = \mu_1 + \cdots + \mu_n$. Let $t \in (0,1)$ be a parameter TBD. We have

$$\begin{aligned} \mathbf{P}\left[\frac{1}{n}(X_1 + \cdots + X_n) \geq \mu + \epsilon\right] &= \mathbf{P}[X_1 + \cdots + X_n \geq \mu_1 + \cdots + \mu_n + \epsilon n] \\ &= \mathbf{P}\Big[e^{t(X_1 + \cdots + X_n)} \geq e^{t(\mu_1 + \cdots + \mu_2 + \epsilon n)}\Big] \\ &\leq \mathbf{E}\Big[e^{t(X_1 + \cdots + X_n - \mu_1 - \cdots - \mu_n - \epsilon n)}\Big] \\ &\overset{\text{(a)}}{=} e^{-\epsilon t n} \prod_{i=1}^{n} \mathbf{E}\Big[e^{t(X_i - \mu_i)}\Big]. \end{aligned}$$

(a) is by the independence of the X_i's. For each i, $X_i - \mu_i$ is bounded in $[-1, 1]$ and has mean 0. By Lemma 13.8, we have

$$\mathbf{E}\Big[e^{t(X_i - \mu_i)}\Big] \leq e^{t^2/2}.$$

Plugging back in, we have

$$\mathbf{P}[\cdots] \leq e^{nt^2/2 - \epsilon t n}.$$

The RHS is minimized by setting $t = \epsilon$, hence

$$\mathbf{P}[\cdots] \leq e^{\epsilon^2 n/2}.$$

□

Chapter 14

PAC learning

In this chapter we discuss foundations of learning theory particularly as it relates to randomized analysis. We start from the *consistent learning model*, which is appealingly simple but does not capture *generalization error*. This leads to the *probably approximately correct (PAC)* learning model, which crucially incorporates randomization to correct this deficiency. From there we explore the notion of generalization error and study its correspondence to concentration inequalities that we have already studied.

14.1 The consistency model

In this chapter we are interested *classification algorithms* that assign labels to unlabeled data, generally based on previous experience analyzing a limited set of labeled training data. Instead of the design and analysis of such algorithms, we focus on developing *models* that allow us to more sensibly declare if such an algorithm is really "learning" or not.

Our first model of learning is called the *consistency* model. Let X denote the space of all possible examples, also called the *domain*. We consider binary classification problems where each $x \in X$ is labeled 0 or 1. A mapping $c : X \to \{0, 1\}$ from examples to labels is called a *concept*. We assume that X is labeled by an unknown concept c from a known collection of concepts $\mathcal{C} \subseteq 2^X$, also called a *concept class*.

The high-level goal is to identify the underlying true concept c given access to a limited set of data. More precisely, the input concepts of a set of labeled examples

$$(x_1, y_1), \ldots, (x_m, y_m)$$

where $x_i \in X$ and $y_i = c(x_i) \in \{0, 1\}$. A concept c' is *consistent* if $c'(x_i) = y_i$ for all i. A *learning algorithm in the consistency model* is one that, given the labeled examples above, either

(a) outputs a concept $c' \in \mathcal{C}$ consistent with the examples, or

(b) decides that no such concept exists.

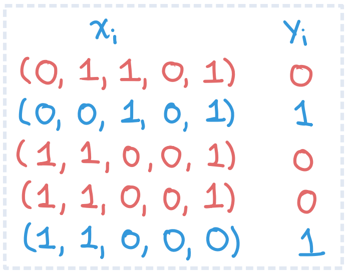
The concept class $\mathcal{C}$ is *learnable in the consistency model* if there is a learning algorithm in the consistency model for it.

Of course we are only interested in *efficient* learning algorithms. The concept class $\mathcal{C}$ is typically extremely large or even infinite, so directly inspecting all concepts in $\mathcal{C}$ is too slow.

It is obviously desirable to be able to identify a concept $c' \in \mathcal{C}$ that agrees with all the labeled input data. However, the consistency model does not ask c' to agree the true concept c beyond the labeled data. This can lead to some undesirable behavior, such as overfitting, as in the following examples.

Example: disjunctions. Recall that a boolean formula $f(x_1, \ldots, x_n)$ is a *disjunction* if it is the disjunction ($\vee$) of x_i's and $\bar{x}_i$'s. Let $X = \{0,1\}^n$, and let $\mathcal{C}$ be the class of disjunctions over n-variables. That is, each $c \in \mathcal{C}$ corresponds to a disjunction $f_c(x_1, \ldots, x_n) = x_{i_1} \vee \bar{x_{i_2}} \vee \cdots \vee \bar{x_{i_k}}$, where $c(x) = 1$ iff $f_c(x) = $ `true`.

Given a set of labeled points $\{(x_i, y_i)\}$ in $\mathbb{R}^d$, we can define a disjunction $f(x_1, \ldots, x_n) = x_{j_1} \vee \cdots \vee x_{j_k}$ where we include:



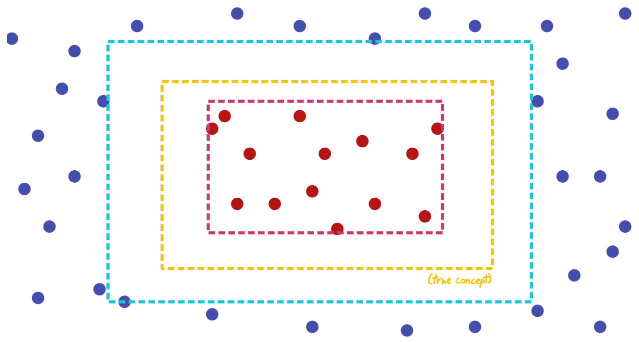
x_i	y_i
(0, 1, 1, 0, 1)	0
(0, 0, 1, 0, 1)	1
(1, 1, 0, 0, 1)	0
(1, 1, 0, 0, 1)	0
(1, 1, 0, 0, 0)	1

(a) Variable x_j iff $x_{i,j} = 0$ for all samples x_i with $y_i = 0$.

(b) Negated variable $\bar{x}_j$ iff $x_{i,j} = 1$ for all samples x_i with $y_i = 1$.

Observe that f will correctly label all of the data in the sample, and the algorithm to construct f is efficient. Thus this concept class is efficiently learnable in the consistency model.

In general, if $f^\star(x_1, \ldots, x_n)$ is the disjunction defining the true concept $c^\star$, then the function f we construct is a "superset" of $f^\star$ in terms of the symbols x_i and $\bar{x}_i$ it contains. Therefore if f is much larger than $f^\star$, then it will overfit the training sample and generally label many more x's as `true` than $f^\star$.



Example: rectangles. Let $X = [0,1]^2$, and let $\mathcal{C}$ be the set of concepts defined by rectangles in $[0,1]^2$. That is, each $c \in \mathcal{C}$ is associated with a rectangle $R_c = [a,b] \times [c,d]$, and $c(x) = 1$ iff $x \in R_c$.

Given a set of labeled points $\{(x_i, y_i) \in [0,1]^2\}$, it is easy to identify the smallest rectangle R_1 containing all (and only) the x_i's labeled $y_i = 1$. Likewise one can identify the largest rectangle R_2 containing the same x_i's. So by outputting either R_1 or R_2 (or anything in between), $(X, \mathcal{C})$ is efficiently learnable in the consistency model.

All we know about the true concept is that it is induced by a rectangle $R^\star$ such that $R_1 \subseteq R^\star \subseteq R_2$. If the difference between R_1 and R_2 is very big (particular when the data is limited and/or not uniformly spread), then neither R_1 nor R_2 may be a very good classifier on real data.

14.2 The PAC learning model

The problem with the consistency model is that it does not incorporate *generalization* beyond the given training examples. This weakness is address in our next model, called the *probably approximately correct (PAC) learning* model and introduced in [Val84a; Val84b].

The PAC learning model is an extension of the consistent learning model. We retain the same setup where X denotes the space of examples, $\{0,1\}$ the set of labels, and $\mathcal{C}$ the concept class. The new ingredients are as follows. First, we assume that the training examples are independently and identically distributed from a fixed (but unknown) distribution $\mathcal{D}$. Second, we define a second family of labeling functions, $\mathcal{H} \subseteq 2^X$, called the *hypothesis class*. (Possibly $\mathcal{H} = \mathcal{C}$.) Our algorithms output a hypothesis $h \in \mathcal{H}$ that, ideally, agrees as much as possible with the unknown concept c. The classification error of a hypothesis h is measured by

$$\operatorname{err}(h) \overset{\text{def}}{=} \mathop{\mathbf{P}}_{x \sim \mathcal{D}}[h(x) \neq c(x)].$$

We now define a *PAC learning algorithm* for $(\mathcal{C}, \mathcal{D})$. Let $\epsilon, \delta \in (0,1)$ be fixed parameters. A PAC learning algorithm takes as input $m = \operatorname{poly}(1/\epsilon, 1/\delta)$ (labeled) samples drawn from $\mathcal{D}$, and produces a hypothesis h. The output hypothesis h is randomized since it reflects a randomized training set; besides, the algorithm itself is allowed to use additional randomization. To meet the criteria of PAC learning, the hypothesis h must have classification error at most ϵ with probability $1 - \delta$. This requirement can be stated compactly as:

$$\mathop{\mathbf{P}}_{h}[\mathrm{err}(h) \leq \epsilon] = \mathop{\mathbf{P}}_{h}\left[\mathop{\mathbf{P}}_{x \sim \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon\right] \geq 1 - \delta. \tag{14.1}$$

(14.1) is a mathematical expression of generalization. Unlike the consistency model of learning, PAC learning does not require h to be particularly accurate on the sample set. (14.1) is only concerned with h's performance with respect to the entire distribution $\mathcal{D}$.

Example: rectangles. Recall the earlier example where the concepts are induced by rectangles in $[0,1]^2$. There we observed that we have efficient consistent learning algorithms by outputting either the smallest rectangle R_1 or largest rectangle R_2 containing all of the positively labeled examples. We also observed that the true rectangle $R^\star$ lied somewhere between R_1 and R_2, which is a weak guarantee if R_1 and R_2 differ greatly.

Suppose $\mathcal{D}$ drew points $x \in [0,1]^2$ uniformly from $[0,1]$. As the number of samples m grows large, we expect points that our closer and closer to each side of the boundary of $R^\star$, and R_1 and R_2 should converge over time. One can show that this concept is PAC learnable by directly calculating the generalization error for either the smallest or largest enclosing rectangle. We will prove this later (perhaps with weaker constants) when we more generally analyze concept classes based on their VC dimension.

14.3 Generalization for finite hypothesis classes

We first analyze the generalization error for a learning algorithm in the consistent learning model, as a function of the number of hypotheses, $|\mathcal{H}|$. In particular these bounds only hold when $\mathcal{H}$ is finite. In the following theorem, we need m to be proportional to $\ln(|\mathcal{H}|)$ to guarantee low generalization error.

Theorem 14.1. *Let $\epsilon, \delta \in (0,1)$ and consider a training set of size m drawn from $\mathcal{D}$. If $m \geq \ln(|\mathcal{H}|/\delta)/\epsilon$, then with probability at least $1 - \delta$, every consistent hypothesis $h \in \mathcal{H}$ has true error $\mathrm{err}(h) \leq \epsilon$.*

Proof. Call a hypothesis $h \in \mathcal{H}$ *good* if $\mathrm{err}(h) \leq \epsilon$, and *bad* if $\mathrm{err}(h) > \epsilon$. It suffices to show that with probability at least $1 - \delta$, all bad hypotheses are inconsistent with the training data.

Fix a bad hypothesis h. For each sample $(x_i, y_i) \sim \mathcal{D}$, we have $\mathbf{P}[h(x_i) = y_i] < 1-\epsilon$. The probability that h agrees on all m independent samples $(x_1, y_1), \dots, (x_m, y_m)$ is

$$(1-\epsilon)^m \leq e^{-\epsilon m} \leq \frac{\delta}{|\mathcal{H}|}.$$

By the union bound, the probability that some bad hypothesis h is consistent is at most

$$\sum_{h \text{ bad}} \mathbf{P}[h \text{ consistent}] \leq |\mathcal{H}| \cdot \frac{\delta}{|\mathcal{H}|} \leq \delta,$$

as desired. □

Uniform convergence. For a training set $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$ and hypothesis h, let

$$\operatorname{err}(h \,|\, S) = \frac{1}{m}|\{i : h(x_i) \neq y_i\}|$$

denote the *empirical or training error rate* of h with respect to S. Most learning algorithms try to select $h \in \mathcal{H}$ minimizing the training error $\operatorname{err}(h \,|\, S)$, in the hopes that it would minimize $\operatorname{err}(h)$ as well. Here we run two kinds from risk when the empirical error $\operatorname{err}(h \,|\, S)$ deviates from the true error $\operatorname{err}(h)$.

1. A very good hypothesis h with low $\operatorname{err}(h)$ may have $\operatorname{err}(h \,|\, S)$ much larger than $\operatorname{err}(h)$, dissuading the algorithm from selecting h.

2. A very bad hypothesis h with high $\operatorname{err}(h)$ may have $\operatorname{err}(h \,|\, S)$ much smaller than $\operatorname{err}(h)$, tempting the algorithm from selecting h.

Thus, for minimizing $\operatorname{err}(h \,|\, S)$ to be a good strategy for minimizing $\operatorname{err}(h)$, we need $\operatorname{err}(h \,|\, S)$ to be close to $\operatorname{err}(h)$ for all the hypotheses h. This is given by the following guarantee.

Theorem 14.2. *Let $\epsilon, \delta \in (0,1)$ and consider a training set S of size m. If $m \geq \ln(2|\mathcal{H}|/\delta)/2\epsilon^2$, then with probability at least $1-\delta$,*

$$|\operatorname{err}(h) - \operatorname{err}(h \,|\, S)| \leq \epsilon$$

for all $h \in \mathcal{H}$.

Proof sketch. We apply the additive Chernoff bound to each $h \in \mathcal{H}$, and take the union bound. □

Example: disjunctions Recall the concept class $\mathcal{C}$ of disjunctions over $X = \{0,1\}^n$. Let $\mathcal{H} = \mathcal{C}$, and recall that we have a consistent learning algorithm for $\mathcal{C}$. We have $|\mathcal{H}| = |\mathcal{C}| = 3^n$ since each disjunction f is defined by selecting, for each $i \in [n]$, whether x_i, $\bar{x}_i$, or neither appears in f. By Theorem 14.2, for $m = O((n + \log(1/\delta))/\epsilon^2)$, the algorithm produces a hypotheses h with $\text{err}(h) \leq \epsilon$ with probability at least $1 - \delta$. That is, for fixed n, the class of disjunctions is PAC-learnable.

14.4 Occam's razor

Occam's razor refers the general principle that one should prefer simpler explanations. It has been echoed by many thinkers in history; for example:

- Aristotle: "Nature operates in the shortest way possible."
- William of Occam: "Plurality is never to be posited without necessity."
- Isaac Newton: "We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances. Therefore, to the same natural effects we must, as far as possible, assign the same causes."
- Bertrand Russell: "Whenever possible, substitute constructions out of known entities for inferences to unknown entities."
- Albert Einstein: "Everything should be made as simple as possible, but not simpler."
- Richard Feynman: "The truth always turns out to be simpler than you thought."
- The Big Aristotle: "If you don't stick to simplicity, you'll die a horrible death."

For learning algorithms, one way to measure the complexity of a hypothesis class $\mathcal{H}$ is via its *descriptive bit complexity*. Fix some system by which hypotheses $h \in \mathcal{H}$ are encoded in bit strings. For a hypothesis $h \in \mathcal{H}$, let $|h|$ denote the number of bits needed to describe h. The *descriptive complexity* of $\mathcal{H}$ is defined as the maximum number of bits, $|h|$, need to describe any hypothesis $h \in \mathcal{H}$. The following gives generalization bounds in terms of the bit complexity of $\mathcal{H}$.

Corollary 14.3. *Let $\mathcal{H}$ be a family of hypotheses of descriptive complexity b. Let $\epsilon, \delta \in (0,1)$ and consider a training set S of size m.*

(i) If $m \geq (b\ln(2) + \ln(1/\delta))/\epsilon$, then with probability at least $1 - \delta$, all consistent hypotheses $h \in \mathcal{H}$ have $\text{err}(h) \leq \epsilon$.

(ii) If $m \geq (b\ln(2) + \ln(2/\delta))/2\epsilon^2$, then with probability at least $1-\delta$, all hypotheses $h \in \mathcal{H}$ have $|\mathrm{err}(h \mid S) - \mathrm{err}(h)| \leq \epsilon$.

Proof. We have $|\mathcal{H}| \leq 2^b$ because b bits can describe at most 2^b hypothesis. □

Regularization by bit complexity. Corollary 14.3 says that hypothesis classes of low bit complexity have better generalization error. Another way to express this theme is to say that every hypothesis $h \in \mathcal{H}$ has generalization error proportional to its bit complexity, as follows.

Corollary 14.4. *Let $\epsilon, \delta \in (0,1)$ and consider a training set of size m drawn from $\mathcal{D}$. With probability at least $1-\delta$, we have*

$$|\mathrm{err}(h) - \mathrm{err}(h \mid S)| \leq O\left(\sqrt{|h| + \ln(1/\delta)/m}\right)$$

for all $h \in \mathcal{H}$.

Proof. For $i \in \mathbb{N}$, let $\mathcal{H}_i$ be the subset of hypothesis of bit complexity between 2^{i-1} and 2^i:

$$\mathcal{H}_i \overset{\text{def}}{=} \left\{h \in \mathcal{H} : 2^{i-1} < |h| \leq 2^i\right\}.$$

Let $\delta_i = \delta/2^i$. By Corollary 14.3, for each $\mathcal{H}_i$, we have

$$\begin{aligned}|\mathrm{err}(h) - \mathrm{err}(h \mid S)| &\leq O\left(\sqrt{\ln(|\mathcal{H}_i|) + \ln(\delta_i)/m}\right)\\ &= O\left(\sqrt{\ln(2^i) + \ln(2^i/\delta)/m}\right)\\ &= O\left(\sqrt{(|h| + \ln(1/\delta))/m}\right)\end{aligned}$$

for all $h \in \mathcal{H}_i$ with probability of error at most δ_i. Taking the union bound over all i, the probability of error over all of $\mathcal{H} = \bigcup_i \mathcal{H}_i$ is $\sum_i \delta_i = \delta \sum_i 1/2^i = \delta$. □

14.5 Stronger generalization bounds via growth rate

Many learning algorithms select from a hypothesis class $\mathcal{H}$ that is geometrically fairly simple. A recurring example has been a simple consistent learner for the concept class of rectangles, which of course generalizes to higher dimensions. The *perceptron algorithm* looks for a hyperplane that fits the data. This algorithm is particularly useful when combined with techniques such as the *kernel trick* that first embed the data into a higher-dimensional space where the labeled data becomes linearly separable.

Recall the previous chapter that the measure of simple geometric objects is particularly well-concentrated under random sampling. Applying those techniques – namely, ϵ-nets and ϵ-samples – to learning gives the following generalization bounds.

To extend the notion of a growth function to a hypotheses class $\mathcal{H}$, we interpret $\mathcal{H}$ as a family of ranges corresponding to the positively labeled sets,

$$\left\{h^{-1}(1) : h \in \mathcal{H}\right\}.$$

Recall that the *growth function* $g(n)$ of $\mathcal{H}$ is the maximum number of distinct subsets that can be induced by $\mathcal{H}$ over a set of n points.

Theorem 14.5. *Let $\mathcal{H}$ be a family of hypotheses with growth function $g(n)$. Let $\epsilon, \delta \in (0,1)$. Consider a data set of size m.*

(i) If $m \geq O(\ln(g(2m)/\delta)/\epsilon)$, then with probability at least $1-\delta$, all consistent hypotheses $h \in \mathcal{H}$ have $\operatorname{err}(h) \leq \epsilon$.

(ii) If $m \geq O\Big(\ln(g(2m)/\delta)/\epsilon^2\Big)$, then all $h \in \mathcal{H}$ have $|\operatorname{err}(h \mid S) - \operatorname{err}(h)| \leq \epsilon$.

Proof. For each hypothesis h, let $barh(x) = 1 - h(x)$ denote the complementary hypothesis. Increasing $|\mathcal{H}|$ and the growth function $g(n)$ by at most a factor of 2 (why?), we assume that that $\mathcal{H}$ contains the complement $\bar{h}$ of each hypothesis $h \in \mathcal{H}$.

Consider the (possibly infinite) point set

$$P = X \times \{0,1\} = \{(x,y) : x \in X,\, y \in \{0,1\}\}$$

We associate each hypothesis $h \in \mathcal{H}$ with the range

$$r_h = \{(x, 1 - h(x))\} \subseteq P.$$

The distribution $\mathcal{D}$ induces a measure $\mu_{\mathcal{D}}$,

$$\mu_{\mathcal{D}}(r_h) = \mathop{\mathbf{P}}_{(x,y)\sim\mathcal{D}}[(x,y) \in r_h]$$

The measure of a range r_h equals the true error of h:

$$\mu_{\mathcal{D}}(r_h) = \mathop{\mathbf{P}}_{(x,y)\sim\mathcal{D}}[(x,y) \in r_h] = \mathop{\mathbf{P}}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] = \operatorname{err}(h).$$

Let $R_{\mathcal{H}} = \{r_h : h \in \mathcal{H}\}$. Let $f(m)$ denote the growth rate of $R_{\mathcal{H}}$. The key claim is that

$$f(m) \leq g^2(n). \tag{14.2}$$

Assuming (14.2) is true, the theorem follows from the ϵ-net and ϵ-sample theorems from Chapter 13, as we now explain.

For result (i), call a hypotheses bad if $\operatorname{err}(h) > \epsilon$; we want to show that for a sample of the claimed size m, with probability at least $1 - \delta$, every bad hypotheses is inconsistent with the data. Here we observe that since $\operatorname{err}(h) = \mu_{\mathcal{D}}(r_h)$, this is equivalent to saying that the sample set is an ϵ-net. The claim now follows from Theorem 13.4.

For result (ii), observe that $\operatorname{err}(h) = \mu_{\mathcal{D}}(r_h)$, and for a training set S, $\operatorname{err}(h \mid S) = \mu_S(r_h)$. Thus result (ii) is equivalent to saying that S is an ϵ-sample with probability at least $1 - \delta$. The sufficient bounds for m now follow from Theorem 13.2.

It remains to prove (14.2). Fix a S set of m labeled points $\{(x_1, y_1), \dots, (x_m, y_m)\}$. Let $S_0' = \{x_i : (x_i, 1) \in S\}$ denote the set of input points labeled 0 and let $S_0 = S_0 \times \{0\}$ denote the corresponding labeled pairs. Likewise let $S_1' = \{x_i : (x_i, 1) \in S\}$ and $S_1 = S_1 \times \{1\}$ be the corresponding sets for label 1.

The total number of intersections induced by $R_{\mathcal{H}}$ on S, $|R_{\mathcal{H}} \wedge S|$, is bounded above by

$$|R_{\mathcal{H}} \wedge S| \leq |R_{\mathcal{H}} \wedge S_0| \cdot |R_{\mathcal{H}} \wedge S_1|,$$

since any two intersections in $|R_{\mathcal{H}} \wedge S|$ must differ on either S_0 or S_1. We claim that both of the terms on the RHS are bounded above by $g(m)$, which proves the claim.

Consider $|R_{\mathcal{H}} \wedge S_0|$. For any $h \in \mathcal{H}$, we have

$$r_h \wedge S_0 = \{(x, 0) \in S : h(x) = 1\} = \left(h^{-1}(1) \wedge S_0'\right) \times \{0\}.$$

That is,

$$|R_{\mathcal{H}} \wedge S_0| = |\mathcal{H} \wedge S_0'| \leq g(m).$$

Similarly, for $|R_{\mathcal{H}} \wedge S_1|$, for each hypothesis h we have

$$r_h \wedge S_1 = \{(x_i, 1) \in S : h(x_i) = 0\} = \left(\bar{h}\ \ ^{-1}(1) \wedge S_1'\right) \times \{1\}.$$

Consequently

$$|R_{\mathcal{H}} \wedge S_1| = |\mathcal{H} \wedge S_0'| \leq g(m).$$

This completes the proof of the claim, hence the proof of the theorem. $\square$

Example: rectangles. Consider the concept class $\mathcal{C}$ of rectangles in $\mathbb{R}^2$. We have a consistent learning algorithm by outputting any rectangle that contains only the positively labeled points of the training set.

Rectangles have constant VC-dimension, hence polynomial growth $g(n) = n^{O(1)}$ (cf. Sauer's lemma, Theorem 13.7). So we need $O(\ln(1/\delta)/\epsilon)$ to guarantee that with probability at least $1 - \delta$, the rectangle we output has true error at most ϵ. That is, rectangles are PAC-learnable.

More generally, any concept class with fixed VC-dimension and a consistent learning algorithm is PAC-learnable.

14.6 Additional notes and materials

These notes are based on the first 10 lectures of [Sch19] and sections 5.1–5.7 of [BHK20]. We refer the interested reader to [Sch19] for additional background on these and other topics in learning theory.

Lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

14.7 Exercises

Exercise 14.1. The definition of PAC learning requires finding a low-error hypothesis with probability at least $1 - \delta$ for aribtrary $\delta > 0$. Consider the following weaker definition of PAC learning where we drop the requirement on δ: let us say that an algorithm is a *weak PAC learner* if for any $\epsilon > 0$, with a training set of size $\text{poly}(1/\epsilon)$, and in randomized polynomial time, it produces a hypothesis with error at most ϵ with probability at least $1/2$. Design and analyze a system that takes as input a weak PAC learning algorithm and produces a PAC learning algorithm (in the original sense).

Chapter 15

Randomized greedy algorithms

15.1 Coverage and submodular functions

In the maximum coverage problem, we have m elements $[m] = \{1, \ldots, m\}$, and n sets $A_1, \ldots, A_n \subseteq [m]$. We also have a cardinality constraint $k \in \mathbb{N}$. The goal is to select (at most) k sets $A_{e_1}, A_{e_2}, \ldots, A_{e_k}$ that maximizes the size of the union

$$|A_{e_1} \cup A_{e_2} \cup \cdots \cup A_{e_k}|.$$

This problem is NP-Hard.

We can interpret coverage as a set function f as follows. We identify a set A_e by its index e. We identify a collections of indices $S \subseteq [n]$ with the corresponding collection of sets $\{A_e : e \in S\}$. We define a function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$ by

$$f(S) \overset{\text{def}}{=} (\text{\# points covered by sets (w/ index) in } S) = \left|\bigcup_{e \in S} A_e\right|.$$

The goal is to maximize $f(S)$ subject to $|S| \leq k$.

Our analysis abstracts out coverage and uses only the following properties of a set function $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$:

- *Normalized:* $f(\emptyset) = 0$.
- *Nonnegativity:* $f(S) \geq 0$ for all S.
- *Monotonicity:* $f(S) \leq f(T)$ if $S \subseteq T$.
- *Submodularity:* For nested sets $S \subseteq T \subseteq \mathcal{N}$, and $e \in \mathcal{N}$,

$$f(S + e) - f(S) \geq f(T + e) - f(T).$$

Submodularity is a natural property both in combinatorics (e.g., coverage, or the rank function of a matroid) and in real-world settings. For example, if we interpret f as a utility function over sets of items, then $f(S+e) - f(S)$ is the marginal gain of adding item e to a collection S. Submodularity says that the marginal gain of e to S is decreasing in S. Economists call this "decreasing marginal returns". Here we introduce a convenient notation for marginal gains:

$$f(A \mid S) \overset{\text{def}}{=} f(A \cup S) - f(S).$$

If f is submodular, then $f(A \mid S)$ is decreasing in A for all sets A (and not just singletons). Note that $g(A) = f(A \mid S)$ defines another set function. Note that if f is submodular, then g is submodular. Similarly for monotonicity. Note that g is automatically normalized, even if f wasn't.

Henceforth we assume that any set function f is always normalized. We assume access to f via an "oracle model" where one can query $f(S)$ for any given value of S.

15.2 Greedy algorithm

The simply greedy algorithm repeatedly takes the item of maximum marginal gain until the cardinality constraint is met. For maximum coverage, this would be:

> *Starting with an empty collection of sets. For k iterations, select the set that covers the most elements left uncovered by the current collection, and that set to the collection.*

More formally, for a set function f:

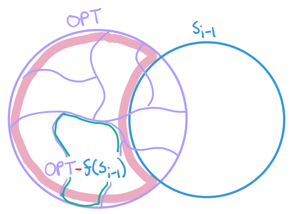
```
greedy($f$,$\mathcal{N}$,$k$)
1. $S_0 \leftarrow \emptyset$.
2. For $i = 1, \ldots, k$:
    A. Let $e_i \in \mathcal{N}$ maximize $f(e_i \mid S_{i-1})$.
    B. $S_i \leftarrow S_i + e$.
3. Return $S_k$.
```

The key lemma. The key lemma analyzes the improvement we make in each iteration. We can interpret Lemma 15.1 below as saying that the greedy algorithm maintains the inequality

$$(\text{improvement}) \geq \frac{1}{k}(\text{room for improvement}),$$

on each iteration.

Lemma 15.1. *For $i = 1, \dots, k$, $f(S_i) - f(S_{i-1}) \geq \frac{1}{k}(\mathrm{OPT} - f(S_{i-1}))$.*



Proof. We first sketch the proof for maximum coverage. There are at least $\mathrm{OPT} - f(S_{i-1})$ elements covered by the optimal solution, but not covered by S_{i-1}. Meanwhile there are k sets in the optimal solution. In particular there is some set in the optimal solution that covers at least $(1/k)$th of the elements. Thus whatever set the greedy algorithm selects adds at least $(1/k)(\mathrm{OPT} - f(S_{i-1}))$ to the total coverage.

For generally monotone submodular f, we have

$$f(S_i) - f(S_{i-1}) = f(e_i \mid S_{i-1}) \overset{(a)}{\geq} \frac{1}{k} \sum_{d \in S^\star} f(d \mid S_{i-1}) \overset{(b)}{\geq} \frac{1}{k} f(S^\star \mid S_{i-1}) \overset{(c)}{\geq} \frac{1}{k}(f(S^\star) - f(S_{i-1})).$$

Here (a) is by greedy choice of e_i. (b) is by submodularity. (c) is by monotonicity. □

Overall analysis. Iterating Lemma 15.1 over k steps leads to the bound.

Theorem 15.2. $f(S_k) \geq (1 - e^{-1})\,\mathrm{OPT}$.

Proof. We have shown that

$$(i\text{th improvement}) = f(S_i) - f(S_{i-1}) \geq \frac{1}{k}(\mathrm{OPT} - f(S_{i-1})) = \begin{pmatrix} \text{room for improvement} \\ \text{after } i-1 \text{ iterations} \end{pmatrix}.$$

Rearranging, we get

$$\mathrm{OPT} - f(S_i) \leq \left(1 - \frac{1}{k}\right)(\mathrm{OPT} - f(S_{i-1})).$$

That is,

$$\begin{pmatrix} \text{room for improvement} \\ \text{after } i \text{ iterations} \end{pmatrix} \leq \left(1 - \frac{1}{k}\right)\begin{pmatrix} \text{room for improvement} \\ \text{after } i-1 \text{ iterations} \end{pmatrix}.$$

We see that "room for improvement" decays exponentially. Unrolling the above, we have

$$\begin{aligned} \mathrm{OPT} - f(S_0) &= \mathrm{OPT} \\ \mathrm{OPT} - f(S_1) &\leq \left(1 - \frac{1}{k}\right)\mathrm{OPT} \\ \mathrm{OPT} - f(S_2) &\leq \left(1 - \frac{1}{k}\right)^2 \mathrm{OPT} \end{aligned}$$

$$\begin{aligned}
\mathrm{OPT} - f(S_3) &\leq \left(1 - \frac{1}{k}\right)^3 \mathrm{OPT} \\
&\vdots \\
\mathrm{OPT} - f(S_k) &\leq \left(1 - \frac{1}{k}\right)^k \mathrm{OPT} \\
&\overset{\text{(a)}}{\leq} e^{-1}\,\mathrm{OPT}\,.
\end{aligned}$$

The last step (a) comes from the inequality $1 + x \leq e^x$ for all x. The final inequality is the claim, up to rearrangement. ☐

15.3 Nonnegative submodular maximization

Suppose f with a nonnegative submodular function, but not monotone. For example, suppose we had a directed graph $G = (V, E)$, and let $f : 2^V \to \mathbb{R}_{\geq 0}$ be the size of the outcut:

$$f(S) = \left|\delta^+(S)\right| \text{ where } S \subseteq V.$$

f is normalized, nonnegative, and submodular, but not monotone.

Suppose we again want to maximize $f(S)$ subject to $|S| \leq k$. The greedy algorithm won't work because f when is not monotone. For example, for $f(S) = |\delta^+(S)|$, if $k = |V|$, the greedy algorithm will take all of the vertices in the graph. But we certainly don't want to do that.

What goes wrong in the analysis? Where did we require monotonicity? The only point where we use monotonicity is in the last step of the proof of Lemma 15.1, where we observed that

$$f(S^\star \mid S_i) = f(S^\star \cup S_i) - f(S_i) \geq f(S^\star) - f(S_i).$$

When f is not monotone, we don't have $f(S^\star \cup S_{i-1}) \geq f(S_{i-1})$. The extra elements from $S^\star$ could hurt the value!

We introduce randomization to introduce this issue. Consider the following lemma.

Lemma 15.3. *Let f be a submodular function, and let $X \subseteq \mathcal{N}$ be a random subset such that for each $e \in \mathcal{N}$, $\mathbf{P}[e \in X] \leq p$, for a fixed parameter $p \in [0,1]$. Then $\mathbf{E}[f(X)] \geq (1-p)f(\emptyset)$.*

For example, if S_i was a *randomized* set where $\mathbf{P}[e \in S_i] \leq 1/2$ for all e_i, then

$$\mathbf{E}[f(S^\star \mid S_i)] = \mathbf{E}[f(S^\star \cup S_i)] - \mathbf{E}[f(S_i)] \geq \frac{1}{2} f(S^\star) - \mathbf{E}[f(S_i)].$$

So either $\mathbf{E}[f(S_i)] \geq \mathrm{OPT}/2$ already (and we are happy), or we at least make some positive progress.

15.3.1 Continuous extensions of f.

We prove Lemma 15.3 from a continuous perspective. A function $F : [0,1]^{\mathcal{N}} \to \mathbb{R}$ is an *extension* of f if

$$F(\mathbb{1}_S) = f(S)$$

for all sets $S \subseteq \mathcal{N}$, where $\mathbb{1}_S$ is the $\{0,1\}$-indicator function.

Perhaps the simplest continuous extension of F is via randomized rounding. The *multilinear relaxation* of f is the function $F_{\text{ML}}(p)$ defined by

$$F_{\text{ML}}(p) = \mathbf{E}[f(S)]$$

where $S \subseteq \mathcal{N}$ is the random set where each element e is sampled with probability p_e. (One can verify that $F_{\text{ML}}(p)$ is linear in each coordinate e; see exercise C.61.)

In general, any mapping from $p \in [0,1]^{\mathcal{N}}$ to a random distribution of set S_p with marginal values p defines a continuous extension $F(p) = \mathbf{E}[S_p]$. The *convex closure* of f is defined by choosing S_p to minimize $\mathbf{E}[S_p]$:

$$F^-(p) = \inf\{\mathbf{E}[f(S)] : S \subseteq \mathcal{N}, \mathbf{P}[e \in S] = p_e \text{ for all } e \in \mathcal{N}\}.$$

For general f this minimization problem may be difficult. But for submodular functions there is a simple solution, due to Lovász. The high-level idea is that since f has decreasing marginal returns then to minimize $f(S)$, we want to maximize the overlap between elements. The overlap is maximized by the correlated randomized set

$$S_t = \{e \in \mathcal{N} : p_e \geq t\}$$

for $t \in [0,1]$ sampled uniformly at random.

Lemma 15.4. $F^-(p) = \mathbf{E}_t[S_t]$

Proof. Recall that a family of sets $\mathcal{F}$ is a *chain* if we can number the sets $\mathcal{F} = \{S_1, \dots, S_k\}$ such that $S_i \subseteq S_{i+1}$ for all i. Observe that the support of S_t is a chain. (Indeed, $S_t \subseteq S_u$ for $u \leq t$.) It is not difficult to see that it is the unique random set with margins p whose support forms a chain.

Of all randomized distributions of sets S with marginal values given by p, choose the distribution maximizing $\mathbf{E}[|S|]$. (The space of such distributions is compact, so such a set exists.) We claim that the support of S is a chain, which implies that $S = S_t$.

If not, then there are two (nonempty) sets A, B in the support of S such that neither $A \subseteq B$ nor $B \subseteq A$. We have

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B). \tag{15.1}$$

Let $p = \min\{\mathbf{P}[S = A], \mathbf{P}[S = B]\}$. Let T be the random set obtained from S by decreasing the probabilities of $T = A$ and $T = B$ both by p, and increasing the probabilities of $T = A \cup B$ and $T = A \cap B$ both by p. Then (15.1) implies that

$$\mathbf{E}[f(T)] \leq \mathbf{E}[f(S)].$$

Furthermore, since $|A|^2 + |B|^2 < |A \cap B|^2 + |A \cup B|^2$ by convexity of $g(x) = x^2$, we have $\mathbf{E}\Big[|T|^2\Big] > \mathbf{E}\Big[|S|^2\Big]$, a contradiction to the choice of S. □

This brings us back to Lemma 15.3.

Proof of Lemma 15.3. Let x be the marginal values of X. Let $S_t = \{e : x_e \geq t\}$ and let $t \in [0, 1]$ be drawn uniformly at random. We have

$$\mathbf{E}[f(X)] \geq F^-(x) = \mathbf{E}[f(S_t)] \geq \mathbf{P}[t > p]\, \mathbf{E}[f(S_t) \,|\, t \geq p] = (1 - p) f(\emptyset).$$

□

15.3.2 Randomized greedy

The following variation of the greedy algorithm introduces randomization to each iteration. Rather than identify the single element e of maximum marginal value, it identifies the top k elements by marginal value, and samples one of them uniformly at random.

```
randomized-greedy($f$,$k$)
1. $S_0 \leftarrow \emptyset$
2. For $i = 1, \dots, k$:
    A. Let $R_i \subseteq \mathcal{N}$ be the $k$ elements $e$ w/ max $f(e \mid S_{i-1})$.
    B. Sample $e_i \sim R_i$ uniformly at random.
    C. $S_i \leftarrow S_{i-1} + e_i$.
3. Return $S_k$.
```

Conditional on S_{i-1}, we have

$$\mathbf{E}[f(S_i)] - f(S_{i-1}) = \frac{1}{k} \sum_{e \in R_i} f(e \mid S_{i-1}) \geq \frac{1}{k} \sum_{e \in S^\star} f(e \mid S_{i-1}) \geq \frac{1}{k} f(S^\star \mid S_{i-1}),$$

just like before.

Let $p = 1 - 1/k$. Each element e has at most a $1/k$ chance of being selected in any given particular iteration. So

$$\mathbf{P}[e \notin S_i] \geq p^i$$

for all i and e. Therefore, by Lemma 15.3, we have

$$\mathbf{E}[f(S_i)] - \mathbf{E}[f(S_{i-1})] \geq \frac{1}{k}\Big(p^{i-1} \operatorname{OPT} - \mathbf{E}[f(S_{i-1})]\Big).$$

Rearranging, we have

$$\mathbf{E}[f(S_i)] \geq \frac{p^{i-1}}{k} \operatorname{OPT} + p \, \mathbf{E}[f(S_{i-1})].$$

Unrolling, we have

$$\begin{aligned}
\mathbf{E}[f(S_1)] &\geq \frac{1}{k} \operatorname{OPT}, \\
\mathbf{E}[f(S_2)] &\geq \frac{p}{k} \operatorname{OPT} + \frac{p}{k} \operatorname{OPT} = \frac{2p}{k} \operatorname{OPT}, \\
\mathbf{E}[f(S_3)] &\geq \frac{p^2}{k} \operatorname{OPT} + \frac{2p^2}{k} \operatorname{OPT} = \frac{3p^2}{k} \operatorname{OPT}, \\
&\vdots \\
\mathbf{E}[f(S_i)] &\geq \frac{p^{i-1}}{k} \operatorname{OPT} + \frac{(i-1)p^{i-1}}{k} \operatorname{OPT} = \frac{ip^{i-1}}{k} \qquad \text{for each } i.
\end{aligned}$$

For $i = k$,

$$\mathbf{E}[f(S_k)] \geq p^{k-1}\,\mathrm{OPT} \geq \frac{1}{e}\,\mathrm{OPT}\,.$$

In conclusion:

Theorem 15.5. `randomized-greedy(`f`,`k`)` *returns a randomized set S with $|S| \leq k$ and $\mathbf{E}[f(S)] \geq \mathrm{OPT}$.*

15.4 Additional notes and materials

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

15.5 Exercise

Exercise 15.1. Prove that the multilinear extension $F_{\text{ML}}(x)$ is multilinear function. (This means that for all fixed x, and elements $e \in \mathcal{N}$, $f(x + \delta 1_e)$ is a linear function in δ over all δ such that $x_e + \delta \in [0, 1]$.)

Exercise 15.2. Recall that the following randomized greedy algorithm gets a $1/e$-approximation for maximizing normalized nonnegative submodular functions subject to a cardinality constraint of k.

```
randomized-greedy($f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$, $\mathcal{N}$, $k$)
1. $S_0 \leftarrow \emptyset$.
2. For $i = 1, \ldots, k$:
    A. Let $R_i \subseteq \mathcal{N}$ be the $k$ elements $e$ with maximum $f(e \mid S_{i-1})$.
    B. Sample $e_i \sim R_i$ uniformly at random.
    C. Set $S_i \leftarrow S_{i-1} + e_i$.
3. Return $S_k$.
```

What if f was also monotone? Analyze the randomized greedy algorithm for normalized monotone submodular functions f.

Exercise 15.3. The deterministic greedy algorithm takes $O(nkQ)$ time where Q denotes an evaluation query to f. For large values of k this may be prohibitively slow. The following algorithm introduces random sampling to try to speed up the algorithm.

```
subsampled-greedy($f$,$\mathcal{N}$,$k$,$\epsilon \in (0,1)$)
1. $S_0 \leftarrow \emptyset$.
2. For $i = 1, \dots, k$:
   A. Let $R \subseteq \mathcal{N}$ sample $n \log(1/\epsilon)/k$ elements independently with
      repetition.
   B. Let $e_i \in R$ maximize $f(e \mid S_{i-1})$.
   C. Set $S_i \leftarrow S_{i-1} + e_i$.
3. Return $S_k$.
```

Analyze `subsampled-greedy` per the following steps.

1. Analyze the expected running time of `subsampled-greedy`.
2. Prove that for all i, conditional on S_{i-1},

$$\mathbf{E}[f(S_i \mid S_{i-1})] \geq \frac{1-\epsilon}{k}(\text{OPT} - f(S_{i-1})).$$

3. Analyze the overall approximation ratio (in expectation) of S_k.

Chapter 16

Entropy and error-correcting codes

16.1 Coding theory

Coding theory concerns the very practical problem of digital communication over imperfect lines of communication. Here we consider one particular model adopted by Claude Shannon when he was working for the American Telephone and Telegraph Company (AT&T) in 1945. In this model, we imagine two locations, A and B. We want to transmit a bit string $x \in \{0,1\}^m$ from point A to point B. While points A and B are connected by some kind of connection, this connection is imperfect. When sending a bit string from point A to point B, every bit gets flipped independently with some probability p. The goal is to reliably communicate bit strings even in the presence of this faulty connection.

It is not impossible to communicate over a bad connection, as anyone who uses a phone knows. Suppose you are calling a friend, and the reception is not very good. You say something. Your friend 'replies, 'sorry, I couldn't hear you". So you say it agian. Your friend again suggests that they didn't understand you. So you say it again and again and again and eventually you start yelling. Ultimately, you are adding *redundancy* to try to communicate your point.

The goal of coding theory is to add enough redundancy to reliably communicate, but otherwise minimize the amount of redundancy. In particular, we have two functions, an *encoder* $\mathcal{C} : \{0,1\}^m \to \{0,1\}^n$ and a *decoder* $\mathcal{D} : \{0,1\}^n \to \{0,1\}^m$. The encoder takes the input message from $x \in \{0,1\}^m$ and maps to to a longer message $\{0,1\}^n$, where $n \geq m$. The encoded message $\mathcal{C}(x)$ is transmitted. On the other end, the decode receivers a corrupted message $y \in \{0,1\}^n$ of $\mathcal{C}(x)$ and decodes it to some $\mathcal{D}(y) \in \{0,1\}^m$. To model the noisy transmission, let $\mathcal{N} : \{0,1\}^n \to \{0,1\}^n$ be a randomized function that flips each bit independently with probability p. We can diagram the entire transmission process as follows.

$$\underset{(\mathbb{R}^m)}{x} \xrightarrow{\text{encode}} \underset{(\mathbb{R}^n)}{\mathcal{C}(x)} \xrightarrow{\text{bits flip w/ prob. } p} \underset{(\mathbb{R}^n)}{\mathcal{N}(\mathcal{C}(x))} \xrightarrow{\text{decode}} \underset{(\mathbb{R}^m)}{\mathcal{D}(\mathcal{N}(\mathcal{C}(x)))}$$

The *rate of transmission* is the ratio

$$\text{rate of transmission} = \frac{m}{n} = \frac{\#\text{ input bits}}{\#\text{ output bits}}.$$

The *average error rate* is the probability

$$\mathbf{P}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x]$$

over the randomness in $\mathcal{N}$ and over $x \in \{0,1\}^m$ chosen uniformly at random. We want high rate of transmission and low average error rate.

The model is reasonable and high rate of transmission is clearly of practical. But is the problem *fundamental*? Are there basic limits or universal laws for coding? Shannon [Sha48] prove that, as the average error rate tends to 0 and n tends larger, the optimum rate of transmission is

$$1 - p\log\left(\frac{1}{p}\right) - (1-p)\log\left(\frac{1}{1-p}\right).$$

The result is not only practical but beautiful, and elevating coding theory as a mathematical subject. The full theorem is as follows:

Theorem 16.1 ([Sha48])**.** *Consider transmission over a noisy channel where each bit is flipped independently with probability $p \in (0, 1/2)$. Let $H(p) = p\log\frac{1}{p} + (1-p)\log\frac{1}{1-p}$.*

1. *For all $\delta > 0$, and n sufficiently large, there is a coding scheme that has transmission rate at least $1 - H(p) - \delta$ and average error rate at most δ.*
2. *For all fixed $\delta > 0$, and n sufficiently large, any coding scheme with transmission rate at least $1 - H(p) + \delta$ has average error rate greater than δ.*

Entropy. The quantity $H(p)$ is defined as follows.

Definition 16.2. *Let $X \in \mathcal{X}$ be a discrete random variable. The* entropy *of X, denoted* $\mathrm{H}(X)$*, is defined as*

$$\mathrm{H}(X) = \sum_{x\in\mathcal{X}} -\mathbf{P}[X = x]\log(\mathbf{P}[X = x]),$$

with the convention that 0/0 = 0, and that $\log$ *denotes the logarithm base 2.*

For $p \in (0,1)$*,* $H(p)$ *is defined as the entropy* $H(X)$ *of the binary variable* $X \in \{0,1\}$ *with* $\mathbf{P}[X = 1] = p$*. That is,*

$$H(p) = p \log\left(\frac{1}{p}\right) + (1-p)\log\left(\frac{1}{1-p}\right).$$

We will do a more thorough investigation of entropy later on in this article. Here we mention some salient points.

Intuitively, entropy measures the uncertainty of a random variable. $H(p)$ is maximized by $p = 1/2$, which is the hardest Bernoulli variable to predict. For discrete random variables X with n outcomes, $H(X)$ is maximized by the uniform distribution, which has entropy $\log(n)$. For two random variables X and Y, the *conditional entropy of* Y *(conditioned) on* X is the expectation, over X, of the entropy of Y conditioned on X:

$$\mathrm{H}(Y \mid X) \overset{\text{def}}{=} \mathop{\mathbf{E}}_{X}[\mathrm{H}(Y) \mid X].$$

If X and Y are independent, then $\mathrm{H}(Y \mid X) = \mathrm{H}(Y)$. Conditional entropy satisfies

$$\mathrm{H}(X,Y) = \mathrm{H}(X) + \mathrm{H}(Y \mid X),$$

where $\mathrm{H}(X,Y)$ is the entropy of the joint random variable (X,Y). Intuitively, knowing the outcome of X should not increase the uncertainty of Y. This is encoded in the *principle of independence*:

$$\mathrm{H}(Y \mid X) \leq \mathrm{H}(Y).$$

This is equivalent to the inequality

$$\mathrm{H}(X,Y) \leq \mathrm{H}(X) + \mathrm{H}(Y),$$

which states that the entropy of (X,Y) is maximized when X and Y are independent.

As an interesting application of entropy, consider a set $S \subseteq [n]$ selected uniformly at random among all subsets of $[n]$ of size at most pn, for fixed $p \in (0,1)$. Let $Y_1, \ldots, Y_n \in \{0,1\}$ be the indicator variables for each element being sampled in S. We have

$$\mathrm{H}(Y_1, \ldots, Y_n) = \mathrm{H}(S) = \log \sum_{i=0}^{pn} \binom{n}{i},$$

where the sum of binomials is the total number of subsets of size at most pn. On the other hand, by the principle of independence,

$$\mathrm{H}(Y_1, \dots, Y_n) \leq \sum_{i=1}^{n} \mathrm{H}(Y_i) = nH(p).$$

Thus

$$\sum_{i=0}^{pn} \binom{n}{i} \leq 2^{H(p)n} = \left(\left(\frac{1}{p}\right)^{1/p} \left(\frac{1}{1-p}\right)^{1/(1-p)} \right)^{n}.$$

16.1.1 Shannon's Upper Bound

In this section, we prove Shannon's upper bound - the first claim in Theorem 16.1. We first state the part that is relevant.

Theorem 16.3. *For all $\delta > 0$, there exists a coding scheme*

$$(\mathcal{C} : \{0,1\}^m \to \{0,1\}^n, \mathcal{D} : \{0,1\}^n \to \{0,1\}^m)$$

that has average error $\leq \delta$ and transmission rate $\geq 1 - H(p) - \delta$.

Proof. Let $n \in \mathbb{N}$ be a large parameter TBD, and let $m = (1 - H(p) - \delta)n$. Rather than propose a specific code, we define $\mathcal{C} : \{0,1\}^m \to \{0,1\}^n$ to be a *uniformly random* function. Define $\mathcal{D} : \{0,1\}^n \to \{0,1\}^m$ by setting $\mathcal{D}(y)$ to be the point x closest to $\mathcal{C}(x)$, breaking ties arbitrarily.

Since $\mathrm{H}(p)$ is continuous, we can choose $\epsilon > 0$ sufficiently small such that

$$|\mathrm{H}((1+\epsilon)p) - \mathrm{H}(p)| \leq \frac{\delta}{2}.$$

We claim the following for each fixed $x \in \{0,1\}^m$.

1. *For sufficiently large n, with probability of error $\leq \delta/2$, no other point $x' \in \{0,1\}^m, x' \neq x$ has its codeword $\mathcal{C}(x')$ within $(1+\epsilon)pn$ bits of $\mathcal{N}(\mathcal{C}(x))$.*

2. *For sufficiently large n, with probability of error $\leq \delta/2$, the noisy transmission $\mathcal{N}(\mathcal{C}(x))$ flips at most $(1+\epsilon)pn$ bits in $\mathcal{C}(x)$.*

Suppose the above holds and let n be sufficiently large. Then with combined probability of error $\leq \delta$, $\mathcal{C}(x)$ is the only point within $(1+\epsilon)pn$ bits of $\mathcal{C}(x)$. That is, when we consider all of the randomness over the random of choice of $x \in \{0,1\}^m$, the random encoding function $\mathcal{C} : \{0,1\}^m \to \{0,1\}^n$, and the noise $\mathcal{N}(\mathcal{C}(x))$, we have

$$\mathop{\mathbf{P}}_{x,\mathcal{C},\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x] \leq \delta. \tag{16.1}$$

Inequality (16.1) above seems similar to the low average error guarantee we seek, except it is averaging over all random codes. We want to prove there exists a single fixed (and no longer random) code $\mathcal{C}$ with low average error, where as the LHS (16.1) is also randomized over all possible codes $\mathcal{C}$. To isolate a specific code $\mathcal{C}$ with average error δ, we rewrite (16.1) as

$$\mathop{\mathbf{E}}_{\mathcal{C}}[\text{average error of } \mathcal{C}] = \mathop{\mathbf{E}}_{\mathcal{C}}\left[\mathop{\mathbf{P}}_{x,\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x]\right] \leq \delta.$$

To dramatically complete the proof: *by the probabilistic method, there exists an encoding $\mathcal{C} : \{0,1\}^m \to \{0,1\}^n$ such that the average error is $\leq \delta$.*

Claim 1. For sufficiently large n, with probability of error $\leq \delta/2$, no other point $x' \in \{0,1\}^m, x' \neq x$ is within $(1+\epsilon)pn$ bits of $\mathcal{N}(\mathcal{C}(x))$.

We prove the claim conditional on $y = \mathcal{N}(\mathcal{C}(x))$; the unconditional claim immediately follows. Fix $y = \mathcal{N}(\mathcal{C}(x))$. Consider any other input point $x' \in \{0,1\}^m$. By construction, $\mathcal{C}(x')$ is selected uniformly at random from $\{0,1\}^n$. Therefore

$$\mathbf{P}[\|\mathcal{C}(x') - \mathcal{C}(x)\|_0 \leq (1+\epsilon)pn] = 2^{-n} \sum_{i=0}^{(1+\epsilon)pn} \binom{n}{i} \leq 2^{(H((1+\epsilon)p)-1)n}.$$

By the union bound, we have

$$\mathbf{P}[\|\mathcal{C}(x') - \mathcal{C}(x)\|_0 \leq (1+\epsilon)pn \text{ for some } x' \neq x] \leq 2^{m+(H((1+\epsilon)p)-1)n}.$$

The RHS is $\leq \delta/2$ iff

$$m \leq (1 - H((1+\epsilon)p))n - 1 - \log(1/\delta),$$

which occurs iff

$$(\text{transmission rate}) = \frac{m}{n} \leq 1 - H((1+\epsilon)p) - O\left(\frac{\log(1/\epsilon)}{n}\right).$$

By choice of ϵ, we have

$$\begin{aligned} 1 - H((1+\epsilon)p) - O\left(\frac{\log(1/\epsilon)}{n}\right) &\geq 1 - H(p) - \delta/2 - O\left(\frac{\log(1/\epsilon)}{n}\right) \\ &\geq 1 - H(p) - \delta \end{aligned}$$

for n sufficiently large. The claim now follows from the choice of m.

Claim 2. *Fix $y \in \{0,1\}^n$. For sufficiently large n, with probability of error $\leq \delta/2$, a noisy transmission $\mathcal{N}(y)$ flips at most $(1+\epsilon)pn$ bits in y.*

We have

$$\lim_{n\to\infty} \mathbf{P}[(\# \text{ bits flipped}) \geq (1+\epsilon)pn] \overset{\text{(a)}}{\leq} \lim_{n\to\infty} e^{-\epsilon^2 pn/2} = 0.$$

where (a) applies the Chernoff inequality. ☐

16.1.2 Lower bounds for Shannon's capacity

Theorem 16.4. *For all fixed $\delta > 0$, and n sufficiently large, any coding scheme with transmission rate at least $1 - H(p) + \delta$ has average error rate greater than δ.*

Proof. Let $(\mathcal{C} : \{0,1\}^m \to \{0,1\}^n, \mathcal{D} : \{0,1\}^n \to \{0,1\}^m)$ be a coding scheme with transmission rate $> 1 - H(p) + \delta$. We claim that this code has error rate $> \delta$. In fact, will show that error rate can be made arbitrarily large for sufficiently large n.

Let $\epsilon > 0$ be a parameter TBD. Let E be the event that $\mathcal{N}(\mathcal{C}(x))$ differs in at least $(1-\epsilon)pn$ bits from $\mathcal{C}(x)$ and no more than $(1+\epsilon)pn$ bits from $\mathcal{C}(x)$. Let $\bar{E}$ be the complementary event.

We have

$$(\text{average correctness rate}) \leq \mathbf{P}\left[\bar{E}\right] + \mathbf{P}_{x,\mathcal{C}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) = x \mid E].$$

For the first term, we have

$$\begin{aligned}\mathbf{P}\left[\bar{E}\right] &= \mathbf{P}[\|\mathcal{N}(\mathcal{C}(x)) - \mathcal{C}(x)\|_0 < (1-\epsilon)pn] + \mathbf{P}[\|\mathcal{N}(\mathcal{C}(x)) - \mathcal{C}(x)\|_0 > (1+\epsilon)pn] \\ &\leq 2e^{-\epsilon^2 pn/3}\end{aligned}$$

by Chernoff bounds. In particular, for fixed $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbf{P}\left[\bar{E}\right] = 0.$$

To bound the second term, for each x, let

$$Y_x = \left\{y \in \mathcal{D}^{-1}(x) : (1-\epsilon)pn \leq \|x - y\|_0 \leq (1+\epsilon)pn\right\}.$$

For fixed x, under event E, Y_x represents all the possible points for $\mathcal{N}(\mathcal{C}(x))$ that would be decoded to x. We have

$$\mathbf{P}_{x,\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) = x \mid E] = \frac{1}{2^m} \sum_{x\in\{0,1\}^m} \sum_{y \in Y_x} \mathbf{P}[\mathcal{N}(\mathcal{C}(x)) = y].$$

For each x, and for each $y \in Y_x$, we have

$$\mathop{\mathbf{P}}_{\mathcal{N}}[\mathcal{N}(\mathcal{C}(x)) = y] \overset{\text{(a)}}{\leq} p^{(1-\epsilon)pn}(1-p)^{(1-(1-\epsilon)p)n} = 2^{-\operatorname{H}(p)n}\left(\frac{1-p}{p}\right)^{\epsilon pn}$$

Here (a) is because because y differs in at least $(1-\epsilon)pn$ bits. We now have

$$\begin{aligned}\mathop{\mathbf{P}}_{x,\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) = x \mid E] &\leq \frac{1}{2^m}\sum_{x \in \{0,1\}^m} e^{-\operatorname{H}(p)n}\left(\frac{1-p}{p}\right)^{\epsilon pn}|Y_x| \\ &\overset{\text{(b)}}{\leq} 2^{(1-\operatorname{H}(p))n-m}\left(\frac{1-p}{p}\right)^{\epsilon pn} \\ &\overset{\text{(c)}}{\leq} 2^{-\delta n}\left(\frac{1-p}{p}\right)^{\epsilon pn} = 2^{\left(\epsilon\log\left(\frac{1-p}{p}\right)-\delta\right)n}.\end{aligned}$$

Here (b) is because the sets Y_x partition $\{0,1\}^n$, so their cardinalities sum to at most 2^n. (c) is by assumption on the transmission rate. For $\epsilon > 0$ sufficiently small, the RHS tends to 0 as $n \to \infty$. □

16.2 Entropy

In this section, we explore the some of the many interesting properties of entropy. We restate the definition for the reader's convenience.

Definition 16.2. *Let $X \in \mathcal{X}$ be a discrete random variable. The* entropy *of X, denoted $\operatorname{H}(X)$, is defined as*

$$\operatorname{H}(X) = \sum_{x \in \mathcal{X}} -\mathbf{P}[X = x]\log(\mathbf{P}[X = x]),$$

with the convention that 0/0 = 0, and that log *denotes the logarithm base 2.*

For $p \in (0,1)$, $H(p)$ is defined as the entropy $H(X)$ of the binary variable $X \in \{0,1\}$ with $\mathbf{P}[X = 1] = p$. That is,

$$H(p) = p\log\left(\frac{1}{p}\right) + (1-p)\log\left(\frac{1}{1-p}\right).$$

For an alternative definition of the entropy of $X \in \mathcal{X}$, let X' be an independent and identically distributed copy of X. Then

$$\operatorname{H}(X) = \mathop{\mathbf{E}}_{X}\left[\log\left(\frac{1}{\mathbf{P}[X' = X]}\right)\right].$$

Put another way, given a discrete random variable $X \in \mathcal{X}$, let us define[1] the *shock* of X, $S_X > 0$, as the random variable that, if $X = x$, takes the value

$$S_X = \frac{1}{\mathbf{P}[X = x]}.$$

Here $\mathbf{P}[X = x]$ refers to the *a priori* probability of X equaling x. Then the entropy of X is

$$\mathrm{H}(X) = \mathbf{E}[\log(S_X)].$$

16.2.1 Concavity and the maximality principle.

Recall that a function $f : [a, b] \to \mathbb{R}$ is *concave* if for all $x, y \in [a, b]$ and $p \in [0, 1]$, we have

$$pf(x) + (1 - p)f(y) \le f(px + (1 - p)y).$$

By induction, we can extend this to finite convex combinations of points. Let $x_1, \ldots, x_n \in [a, b]$ and $p_1, \ldots, p_n \ge 0$ with $p_1 + \cdots + p_n = 1$. Then we have

$$p_1 f(x_1) + \cdots + p_2 f(x_n) \le f(p_1 x_1 + \cdots + p_n x_n). \tag{16.2}$$

Let $f : [a, b] \to \mathbb{R}$ be a function and let $X \in [a, b]$ be a discrete random variable taking on a finite number of values. Say X takes on n values $x_1, \ldots, x_n$ with probabilities $p_1, \ldots, p_n$ respectively. Then (16.2) is the same as saying that

$$\mathbf{E}[f(X)] \le f(\mathbf{E}[X]).$$

We can extend this to continuous distributions of X by approximation by finite distributions. Thus we have *Jensen's inequality*, which is basically rewriting the definition of concavity.

Lemma 16.5. *Let $X \in [a, b]$ be a random variable, and let $f : [a, b] \to \mathbb{R}$ by concave. Then*

$$\mathbf{E}[f(X)] \le f(\mathbf{E}[X]).$$

Proof. Suppose X takes on only two values, a and b, with probability p and $(1 - p)$ respectively. Then

$$\mathbf{E}[f(X)] = pf(a) + (1 - p)f(b) = f(pa + (1 - p)b) = f(\mathbf{E}[X]).$$

We can extend the argument to any finite number of values by induction. □

[1] This is not a standard terminology.

Lemma 16.6. *Over all discrete distributions over n values, entropy is maximized by the uniform distribution, which has entropy* $\log(n)$.

Proof. Suppose X takes on at most n different values x. Then

$$\mathrm{H}(X) = \mathbf{E}[\log(S_X)] \overset{\text{(a)}}{\leq} \log(\mathbf{E}[S_X]) \overset{\text{(b)}}{=} \log(|\mathcal{X}|)$$

(a) is by Jensen's inequality. (b) is because

$$\mathbf{E}[S_X] = \sum_x \frac{\mathbf{P}[X = x]}{\mathbf{P}[X = x]} = n.$$

On the other hand, if X is the uniform distribution over n values $x_1, \dots, x_n$, then

$$H(x) = \sum_i \mathbf{P}[X = x_i] \log\left(\frac{1}{X = x_i}\right) = \sum_i \frac{1}{n} \log(n) = \log(n).$$

□

16.2.2 Conditional entropy

Let (X, Y) be jointly distributed random variables. Conditional on X, Y is a random variable with a well defined entropy $H(Y)$ (given X).

Definition 16.7. *The conditional entropy of Y on X is defined as*

$$\mathrm{H}(Y \mid X) = \mathop{\mathbf{E}}_X[\mathrm{H}(Y) \mid X]$$

In terms of "shocks", we let $S_{Y|X}$ be the shock value of the conditional variable Y given X. To be precise, conditional on $X = x$ and $Y = y$, $S_{Y|X}$ takes the value

$$S_{Y|X} = \frac{1}{\mathbf{P}[Y = y \mid X = x]},$$

where $\mathbf{P}[Y = y \mid X = x]$ is the *a priori* probability given only $X = x$. Then

$$\mathrm{H}(Y \mid X) = \mathop{\mathbf{E}}_X\left[\mathop{\mathbf{E}}_Y\left[\log\left(S_{X|Y}\right)\right]\right] = \mathop{\mathbf{E}}_{X,Y}\left[\log\left(S_{X|Y}\right)\right]$$

We have the following identity that breaks the joint entropy into two entropy terms.

Lemma 16.8. $\mathrm{H}(X, Y) = \mathrm{H}(Y \mid X) + \mathrm{H}(X)$

Proof. Observe that conditional on $X = x$ and $Y = y$, we have

$$S_{X,Y} = \frac{1}{\mathbf{P}[X = x, Y = y]} = \frac{1}{\mathbf{P}[X = x]\,\mathbf{P}[Y = y \mid X = x]} = S_X S_{Y|X}.$$

Thus

$$\mathrm{H}(X, Y) = \mathop{\mathbf{E}}_{X,Y}[\log(S_{X,Y})] = \mathop{\mathbf{E}}_{X,Y}\Big[\log\Big(S_{Y|X}\Big) + \log(S_X)\Big] = \mathrm{H}(Y \mid X) + \mathrm{H}(X),$$

as desired. □

16.3 Principle of Independence

The following lemma observes that taking conditioning on another variable can only decrease the entropy. This is called the *principle of independence.*

Lemma 16.9. *Let* (X, Y) *be jointly distributed. Then* $\mathrm{H}(Y \mid X) \leq \mathrm{H}(Y)$

Proof. We have

$$\begin{aligned}\mathrm{H}(Y \mid X) &= \mathop{\mathbf{E}}_{X,Y}\Big[\log\Big(S_{Y|X}\Big)\Big] = \mathop{\mathbf{E}}_{Y}\Big[\mathop{\mathbf{E}}_{X}\Big[\log\Big(S_{Y|X}\Big) \,\Big|\, Y\Big]\Big] \\ &\overset{(a)}{\leq} \mathop{\mathbf{E}}_{Y}\Big[\log\Big(\mathop{\mathbf{E}}_{X}\Big[S_{Y|X} \,\Big|\, Y\Big]\Big)\Big] \overset{(b)}{\leq} \mathrm{H}(Y).\end{aligned}$$

(a) is by Jensen's inequality. (b) is because, conditional on $Y = y$, we have

$$\mathop{\mathbf{E}}_{X}\Big[S_{Y|X} \,\Big|\, Y\Big] = \sum_x \frac{\mathbf{P}[X = x \mid Y = y]}{\mathbf{P}[Y = y \mid X = x]} \overset{(c)}{=} \sum_x \frac{p(x)}{p(y)} = \frac{1}{p(y)}.$$

(c) substitutes Bayes' law:

$$\mathbf{P}[X = x \mid Y = y]\,\mathbf{P}[Y = y] = \mathbf{P}[X = x, Y = y] = \mathbf{P}[Y = y \mid X = x]\,\mathbf{P}[X = x].$$

□

16.3.1 What is entropy, really?

We close with a quote from Shannon [TM71].

> My greatest concern was what to call it. I thought of calling it "information", but the word was overly used, so I decided to call it "uncertainty". When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, "You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage".

16.4 Additional notes and materials

For additional background on entropy, see [Gal14].

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

16.5 Exercises

Exercise 16.1. In Section 16.1, we discussed coding schemes for achieving low *average error rates*. Recall that an average error rate of δ means that the transmission failure probability is averaged over all $x \in \{0,1\}^m$:

$$\text{average error} = \mathop{\mathbf{E}}_{x}\left[\mathop{\mathbf{P}}_{\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x)))] \neq x\right] = \mathop{\mathbf{P}}_{x,\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x].$$

By contrast, a *uniform error* of δ means that for *every* $x \in \{0,1\}^m$, the transmission failure probability is at most δ: that is,

$$\text{uniform error} \overset{\text{def}}{=} \max_{x} \mathop{\mathbf{P}}_{\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x] \leq \delta.$$

Prove Shannon's upper bound (Theorem 16.1) for uniform error instead of average error. That is, show that for all fixed $\delta > 0$, there exists a coding scheme $(\mathcal{C} : \{0,1\}^m \to \{0,1\}^n, \mathcal{D} : \{0,1\}^n \to \{0,1\}^m)$ (for m, n sufficiently large) with uniform error δ and transmission rate at least $1 - \mathrm{H}(p) - \delta$.[2]

Exercise 16.2. In Section 16.1, we develop *redundant* codes that are extremely efficient w/r/t their transmission rate. Another problem, moving in sort of the opposite direction, is *compression*.

Here we consider compression in the following model. Let Σ be a finite alphabet of n letters. Our goal is to efficiently assign bit strings (codes) to each letter in Σ

[2]Hint: Given a code with average error rate δ, how many of the input words x have transmission failure probability greater than 2δ?

so that messages, composed of sequences of letters in Σ, are as efficient as possible. More specifically, we are only allowed to use *prefix-free codes*, which are mappings

$$\mathcal{C} : \Sigma \to \{0, 1\}^*$$

assigning bit strings (of varying length) to letters such that no code $\mathcal{C}(x)$ ($x \in \Sigma$) is a prefix to another code $\mathcal{C}(y)$ ($y \in \Sigma$). Prefix codes are particularly easy to decode. As we scan the bits of an encoded message, as soon as we see a string that matches the code of a letter, we immediately decode that the scanned bits to the letter. We then continue to scan the rest of the bits as the beginning of the code of a new letter.

For example, the most straight forward prefix code would be to assign each letter in Σ a different bit string with $\lceil \log n \rceil$ bits.

Prefix codes can be identified with binary trees where each branch represents a 0 or 1 bits, and the leaves correspond to a letter where the root to leaf path gives the encoding of a letter.

We assume that not all letters in Σ are distributed frequently. (This is where we have some opportunity for compression) Let $p \in \Delta^{\Sigma}$ be a fixed distribution over Σ. For each letter in $x \in \Sigma$, p_x represents the average frequency of the letter x in these messages.

Given an encoding $\mathcal{C} : \Sigma \to \{0, 1\}^*$, the average number of bits per letter is

$$\sum_{x \in \Sigma} p_x |\mathcal{C}(x)|,$$

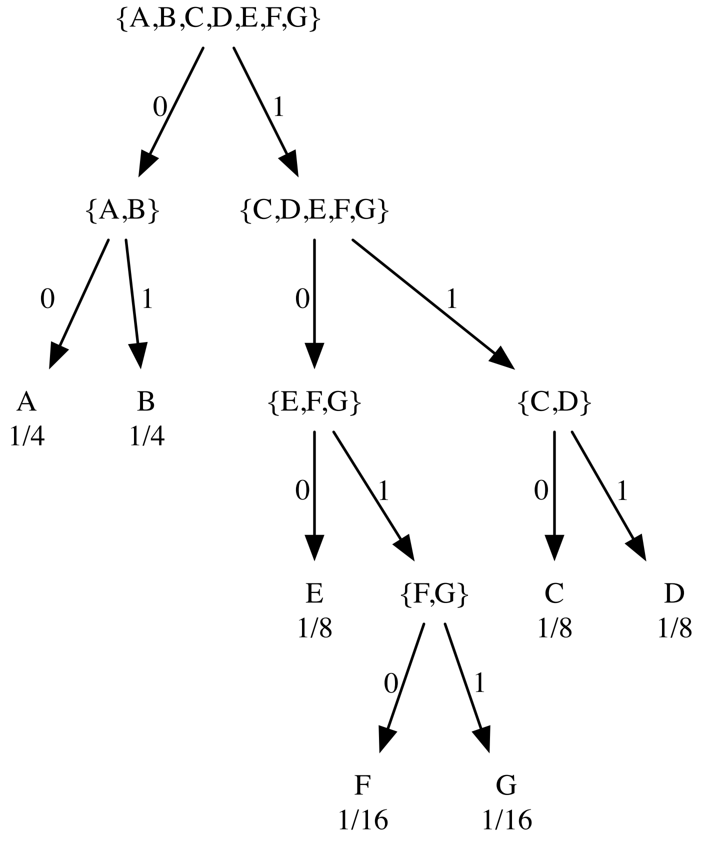
where $|\mathcal{C}(x)|$ denotes the length of the bit string $\mathcal{C}(x)$.

For this problem, consider a special case where every probability p_x is a power of 2, of the form $1/2^{i_x}$ for some integer $i_x \in \mathbb{N}$. Show that there exists a prefix code $\mathcal{C} : \Sigma \to \{0, 1\}^*$ where the average length is *exactly* $H(p)$, where $H(p)$ is the entropy of the random letter drawn from Σ in proportion to p:

$$H(p) = \sum_{x} p(x) \log\left(\frac{1}{p(x)}\right).$$

Here's a harder follow up question: can one do better than the entropy $H(p)$? Either prove that $H(p)$ is optimal, or give a counter example where the probabilities are powers of 2 and one can achieve better than $H(p)$ average bits per letter.

As an example, the following tree defines a prefix code over a distribution of 7 letters $\{A, B, C, D, E, F, G\}$ with probabilities $\{1/4, 1/4, 1/8, 1/8, 1/8, 1/16, 1/16\}$, respectively. One can see that the average length matches the entropy of the distribution.


{A,B,C,D,E,F,G}
0
1
{A,B}
{C,D,E,F,G}
0
1
0
1
A
1/4
B
1/4
{E,F,G}
{C,D}
0
1
0
1
E
1/8
{F,G}
C
1/8
D
1/8
0
1
F
1/16
G
1/16

Chapter 17

Lovász local lemma and resampling

17.1 Resampling k-SAT

Recall that in the k-SAT problem, we are given a CNF formula $f(x_1, \ldots, x_n)$ with m clauses $C_1, \ldots, C_m$ and k (distinct) variables per clause. In max k-SAT, the goal is to satisfy as many clauses as possible, and previously we showed that a random assignment satisfies $(1 - 2^{-k})m$ clauses in expectation. More precisely we showed that each clause is satisfied with probability $1 - 2^{-k}$.

Now suppose $m < 2^k$. By the union bound, we have

$$\mathbf{P}[f(x_1, \ldots, x_n) = \texttt{false}] \leq \sum_{i=1}^{m} \mathbf{P}[i\text{th clause is unsatisfied}] = \frac{m}{2^k} < 1.$$

Therefore, f must be satisfiable!

Of course, when $m > 2^k$, a simple union bound does not show that f is satisfiable. In this chapter, we explore a probabilistic technique called the *local lemma*, that offers an alternative criteria for proving that a k-SAT formula is satisfiable, *independent of m.*

The theorem we unveil below is constructive and comes with a simple randomized algorithm for k-SAT which we now describe. The algorithm is called the *(randomized) resampling algorithm* and was described by [Mos09]. Given a k-SAT formula $f(x_1, \ldots, x_n)$, we first assign $x_1, \ldots, x_n \in \{\texttt{true}, \texttt{false}\}$ independently and uniformly at random. Then, as long as there is a clause C that is unsatisfied, we resample all the variables in C independently and uniformly at random. The algorithm continues to resample unsatisfied clauses until all clauses are satisfied. (*A priori* the algorithm may never terminate.)

Surprisingly, under certain conditions independent of m and n, and which are easy to verify, f always has a satisfying assignment, and the resampling algorithm described above terminates in polynomial time in expectation. Below, we say that two clauses C_i and C_j of f *intersect* if there is a variable x_k appearing (as is, or negated) in both C_i and C_j.

Theorem 17.1. *Let $f(x_1, \dots, x_n)$ be a k-SAT CNF with m clauses, such that each clause intersects at most d other clauses. If $d \leq 2^k/e - 1$, then f is satisfiable. Moreover, a satisfying assignment can be computed by the resampling algorithm while sampling at most m/d clauses in expectation.*

The most striking feature of Theorem 17.1 is that the criteria for satisfiability — $d \leq 2^k/e - 1$ where d bounds the number of clauses overlapping with a single clause — is local. It is easy to inspect a CNF formula and compute d and see if Theorem 17.1 applies.

17.2 The Lovász Local Lemma

The k-SAT result described above is just one example of a more general technique called the *Lovász local lemma (LLL)* [EL75][1]. It applies in a more abstract setting where we have a family of events we want to avoid, and assuming dependency among the events is limited in a particular way.

Formally, let $A_1, \dots, A_n$ denote events in an arbitrary probability space. We write $A_i \sim A_j$ when A_i and A_j are not mutually independent, and $A_i \nsim A_j$ otherwise. The question is whether it is possible for *none* of the events $A_1, \dots, A_n$ to occur simultaneously; i.e., if $\mathbf{P}[\bar{A}_1, \dots, \bar{A}_n] > 0$. [EL75] gave the following sufficient condition which depends only on the *local dependencies* of each event A_i to other events A_j.

Theorem 17.2 ([EL75])**.** *Suppose there exists values $x(A_1), \dots, x(A_n) \in (0,1)$ such that for all events A_i,*

$$\mathbf{P}[A_i] \leq x(A_i) \prod_{\substack{A_j \sim A_i \\ A_j \neq A_i}} (1 - x(A_j)). \tag{17.1}$$

Then $\mathbf{P}[\bar{A}_1, \dots, \bar{A}_n] > 0$.

A simpler ("symmetric") version of Theorem 17.2 that applies to the k-SAT problem is as follows.

Corollary 17.3. *Suppose that each event A_i is mutually independent of all but at most d other events A_j, and $\mathbf{P}[A_i] \leq p$ for all events A_i. If $ep(d+1) \leq 1$, then $\mathbf{P}[\bar{A}_1, \dots, \bar{A}_n] > 0$.*

[1] Erdös attributes the technique to Lovász and helped popularize the name. LLL is also referred to as just the *local lemma* in the literature.

Proof. The claim follows from Theorem 17.2 for $x(A_i) = 1/(d+1)$. □

This establishes the fact that any k-SAT formula with at most $2^k/e$ dependencies per clause is satisfiable. Theorem 17.2 has many other applications both in computer science and combinatorics, such as in proving extremal graph coloring bounds (e.g., [AS16; MR02]). For k-SAT, however, it does not provide an algorithm for actually computing such an assignment.

Developing an *algorithmic LLL* was an important open question. Progress on this question starts in 1991 when [Bec91] analyzed an algorithm for a hypergraph 2-coloring problem that can also be addressed by LLL. Subsequent developments followed in [Alo91; MR98; CS00; Sri08].

In 2009, a breakthrough by Moser [Mos09] proved Theorem 17.1 up to a constant factor in the bound for d. This work proposed the simple resampling algorithm described above and also a surprising proof technique now called the "entropy compression argument". This was followed up by [MT10] which extended the techniques to the more general setting of Theorem 17.2 (via a related but different proof), and obtained tight bounds.

To present the constructive version of Theorem 17.2 we assume a more concrete setting where the events are driven by an underlying set of independent random variables. Let $\mathcal{V}$ be a finite collection of mutually independent random variables in a fixed probability space. Let $A_1, \ldots, A_n$ be a set of events where each event A_i is determined by a subset of variables $S_i \subseteq \mathcal{V}$. (For k-SAT, $\mathcal{V}$ corresponds to the boolean variables $x_1, \ldots, x_n$, and each clause C_i is associated with the event A_i where C_i is not satisfied.) The concrete goal is to compute a configuration of $\mathcal{V}$ so that none of the events $A_1, \ldots, A_n$ occur.

The resampling algorithm extends to this more general setting as follows. We initially sample each variable $X \in \mathcal{V}$ independently from their respective distributions. While there is an event A_i induced by the current sample of variables, we resample the variables in S_i. The algorithm terminates when none of the events $A_1, \ldots, A_n$ occur.

The following theorem from [MT10] states that the resampling algorithm gives an algorithmic LLL.

Theorem 17.4. *Given the setting of Theorem 17.2, the resampling algorithm computes an assignment of values to $\mathcal{V}$ inducing $\bar{A}_1, \ldots, \bar{A}_n$, while resampling each event A_i at most $x(A_i)/(1 - x(A_i))$ times in expectation.*

The rest of this chapter is focused on proving Theorems 17.2 and 17.4 in the more concrete setting with variables $\mathcal{V}$. We follow the proof of [MT10]. We mention that

Theorem 17.2 can be proven without the existence of $\mathcal{V}$ and we refer to [AS16, §5.1] for this proof (which is shorter).

17.3 Analysis of the resampling algorithm

The high-level goal is to bound the total number of resampling steps taken by the resampling algorithm, in expectation.

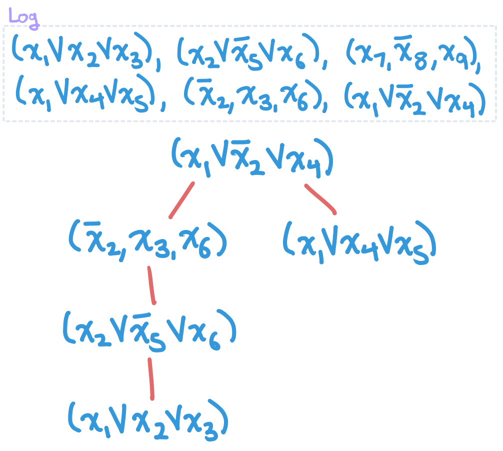
Recall that the algorithm selects events to sample in arbitrary order. For the sake of analysis we fix any deterministic or randomized mechanism to select these events. Each instance of the resampling algorithm can be associated with a *log* where we list the sampled events in chronological order. (E.g., A_5, A_7, A_{11},A_5, ...) Note that a single event may occur multiple times in the log. Let $L \in \mathbb{Z}_{\geq 0} \cup \{\infty\}$ denote the (randomized) length of the log. We want to bound the expected length of the log, $\mathbf{E}[L]$.

Witness trees. Moser and Tardos's [MT10] argument is based on analyzing auxiliary rooted trees called *witness trees*. A *witness tree* is defined as a (nonempty) rooted tree where each node is labeled by an event A_i. We only consider witness trees with the following additional properties.

(a) If a node labeled A_i is a child of a node labeled A_j, then $A_i \sim A_j$.

(b) If two nodes labeled A_i and A_j have the same depth, then $A_i \nsim A_j$.

For each prefix of the log, containing (say) i entries, we associate a witness tree T_i constructed as follows. Let $A_{i_1}, \ldots, A_{i_i}$ be the event labels of the first i entries in the log, where $i \leq L$. We process the events in reverse order. First, we create a root labeled by A_{i_i}. Then, for $j = i - 1$ down to 1, if A_{i_j} shares variables with an event labeling of a node N in the tree, then N be the node of greatest depth, and create a new child of N labeled by A_{i_j}. (If A_{i_j} is independent of all nodes in the tree, then we do not create a node for A_{i_j}.)


Log
$(x_1 \vee x_2 \vee x_3)$, $(x_2 \vee \bar{x}_5 \vee x_6)$, $(x_7, \bar{x}_8, x_9)$,
$(x_1 \vee x_4 \vee x_5)$, $(\bar{x}_2, x_3, x_6)$, $(x_1 \vee \bar{x}_2 \vee x_4)$
$(x_1 \vee \bar{x}_2 \vee x_4)$
$(\bar{x}_2, x_3, x_6)$
$(x_1 \vee x_4 \vee x_5)$
$(x_2 \vee \bar{x}_5 \vee x_6)$
$(x_1 \vee x_2 \vee x_3)$

This process produces a rooted tree with at most i nodes. We see that the tree satisfies property (a) above because a node with label A_i is made a child of a node with label A_j only if A_i and A_j share at least one variable. Property (b) follows from the fact that when introducing a node n with label A_i, we always make n a child of the node n' with maximum depth among those labeled by an event A_j that shares a variable with A_i.

We highlight one observation about this construction that will be particularly useful later on.

Observation 17.5. *Consider a prefix of the log with witness tree T. Let e_1 and e_2 be two entries in the log with e_1 listed before e_2, labeled by events A_{i_1} and A_{i_2}, respectively. If $A_{i_1} \sim A_{i_2}$, and e_2 is a node in the witness tree T, then e_1 is also a node in the witness tree at depth strictly greater than e_1.*

For $i = 1, 2, \ldots$, when $i \leq L$, let T_i be the tree certificate associated by the first i events $A_1, \ldots, A_i$ in the log. For indices $i > L$, we let T_i denote a null value $\emptyset$ (not equal to any witness tree). We can express the expected length of the log in terms of the probabilities of each tree arising at each prefix of the log, as follows:

$$\mathbf{E}[L] = \sum_{i \in \mathbb{N}} \sum_{T} \mathbf{P}[T_i = T], \tag{17.2}$$

where the inner sum is over all certificate trees.

Lemma 17.6. *For all distinct $i, j \in [L]$, $T_i \neq T_j$.*

Proof. Let $i < j$. To have $T_i = T_j$, the ith and jth entry of the log must correspond to the same event A_k.

Suppose that is the case. Then every node added to T_i, including the node corresponding to the ith entry, will also be added to T_j. Since T_j also includes a node for the jth entry, this shows that $|T_j| > |T_i|$, hence $T_j \neq T_i$. □

The fact the all T_i's are distinct allows us to simplify (17.2) to

$$\mathbf{E}[\mathrm{L}] \leq \sum_{T} \mathbf{P}[T = T_i \text{ for some } i].$$

Next we analyze the probability that a fixed wtiness tree T arises (anywhere) in the log. For a witness tree T, let

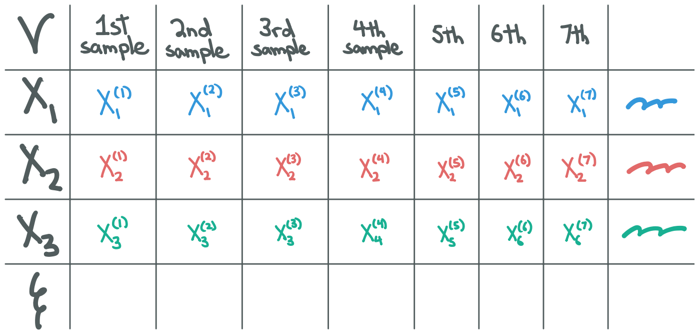
$$p(T) = \prod_{\text{labels } A_i \text{ in } T} \mathbf{P}[A_i],$$

where the product ranges over all events A_i labeling nodes in T, with multiplicity.

Lemma 17.7. *Let T be a fixed witness tree. The probability that T appears in the log, $\mathbf{P}[T_i = T$ for some $i]$, is at most $p(T)$.*

Proof. Consider the following random procedure which we call *checking* T. Let $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ list the labels of T in decreasing order of depth (starting with a leaf, and ending at the root). For each A_{i_j} in order, we resample S_{i_j}. We say that this procedure of checking T *passes* if for each A_{i_j}, resampling S_{i_j} induces A_{i_j}.

The probability that T passes the check is exactly $p(T)$. We want to show that the probability of T occurring in the log is bounded above by the probability that T passes the check.



$\mathcal{V}$	1st sample	2nd sample	3rd sample	4th sample	5th	6th	7th	
X_1	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$	$X_1^{(4)}$	$X_1^{(5)}$	$X_1^{(6)}$	$X_1^{(7)}$	〰
X_2	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$	$X_2^{(4)}$	$X_2^{(5)}$	$X_2^{(6)}$	$X_2^{(7)}$	〰
X_3	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$	$X_4^{(4)}$	$X_5^{(5)}$	$X_6^{(6)}$	$X_6^{(7)}$	〰
ξ								

Recall that the algorithm resamples the variables $\mathcal{V}$ repeatedly, and each variable is sampled independently. We imagine generating, for each variable, an entire (infinite) sequence of samples ahead of time, and then running the resampling algorithm on these fixed sequences of samples. Here the first sample of each variable is used for the initialization. In this scheme, we associate each log-entry e, corresponding to resampling for an event A_i, the set of sampled values for S_i *before* resampling that induced A_i. The sets of sampled values associated to each log entry are disjoint.

Now fix a set of samples which induces a particular log. Whenever T occurs on the log, we check T using the same fixed set of samples. That is, when the check processes A_{i_j} and resamples S_{i_j}, for each variable in S_{i_j}, we use the next sample for that variable not used from processing other events previously in the check. We will show that T occurs in the log only if T passes the check (with respect to the fixed sample). This implies that the probability that T occurs in the log is bounded above by the probability that T passes the check, and completes the proof.

Fix a node N in the tree T which corresponds to a fixed entry e in the log. Let A_i be the label of N and e, and let $X \in S_i$ be any variable in its support. We want to show that both the value for X associated with the event e, and the value of X drawn by the checking procedure when processing N, corresponds to the same sample of X. Now, the resampling procedure uses the jth sample of X where j is the number of entries in the log, up to and including A_i, labeled by an event that depends on X. The tree check uses the kth sample of X where k is the number of events chronologically before and including A_i in the checking process that has X in its support.

We want to show that $j = k$. If this is true for all variables $X \in A_i$, then since the sample corresponding to log entry e induced A_i, the checking procedure will also induce A_i when processing the node n. Taken over all nodes n, we conclude that the witness tree T passes the check.

To show that $j = k$, consider any log entry e' that (a) comes before e and (b) is labeled by an event A_j that also depends on X. As observed above, the construction of the tree certificate implies that there is also a node N' corresponding to e', and N' has depth strictly greater than N. Meanwhile any entry e' that (a) comes *after* e and (b) is labeled by an event depending on X is either not in the tree or appears at a depth strictly less than n. This implies that $j = k$ in the sense described above, and completes the proof. □

It remains to bound the sum of $p(T)$ over all witness trees T. We break this down by the event labeling the root.

Lemma 17.8. *Let $\mathcal{T}_i$ be the family of witness trees with root labeled by A_i. Then*

$$\sum_{T \in \mathcal{T}_i} p(T) \leq \frac{x(A_i)}{1 - x(A_i)}.$$

Proof. Consider the following Galton-Watson branching process producing a tree in $\mathcal{T}_i$.

In the first round we introduce a root labeled A_i. Each round, for each node N introduced in the previous round labeled by an event A_j, we do the following. For each event A_k with $A_k \sim A_j$ (including $A_k = A_j$), with probability $x(A_k)$, we introduce a child of N labeled by A_k.

The process terminates when no new vertices are introduced in a single round. (The process may continue indefinitely).

Fix a particular tree $T \in \mathcal{T}_i$. Let q_T denote the probability that T is produced by the Galton-Watson process. For each node $N \in T$, let A_N denote the event labeling n. Let $\mathcal{B}_N$ denote the set of events A_k such that $A_k \sim A_N$, but the process did *not* produce a child of N with label A_k. We have

$$q_T = \frac{1}{x(A_i)} \prod_{N \in T} x(A_N) \prod_{A_k \in \mathcal{B}_N} (1 - x(A_k)).$$

Here, $\prod_{N \in T} x(A_N)/x(A_i)$ is the probability of sampling all the nodes that are in the tree, and $\prod_{N \in T} \prod_{A_k \in \mathcal{B}_N} (1 - x(A_k))$ is the probability of not sampling all the potential nodes that were left out of the tree. For this second term, we have

$$\prod_{N \in T} \prod_{A_k \in B_N} (1 - x(A_k)) = (1 - x(A_i)) \frac{\prod_{N \in T} \prod_{A_k : A_k \sim A_j} (1 - x(A_k))}{\prod_{N \in T} (1 - x(A_k))}$$

since every A_k omitted from B_N appears as the label of another distinct node, and these omitted events cover all the nodes except the root. This leads to the cleaner bound,

$$\begin{aligned} q_t &= \frac{1 - x(A_i)}{x(A_i)} \prod_{N \in T} \frac{x(A_N)}{1 - x(A_N)} \prod_{A_k \sim A_N} (1 - x(A_k)) \\ &= \frac{1 - x(A_i)}{x(A_i)} \prod_{n \in T} x(A_N) \prod_{\substack{A_k \sim A_N \\ A_k \neq A_N}} (1 - x(A_k)). \end{aligned}$$

Now, by plugging in the inequality (17.1) assumed in the theorem, we have

$$p(T) = \prod_{\text{node } N \in T} \mathbf{P}[A_N] \leq \prod_{n \in T} x(A_N) \prod_{\substack{A_j \sim A_N \\ A_j \neq A_N}} (1 - x(A_j)) = \frac{x(A_i)}{1 - x(A_i)} q_t.$$

Summing over all $T \in \mathcal{T}_i$, we have

$$\sum_{T \in \mathcal{T}_i} p(T) \leq \frac{x(A_i)}{1 - x(A_i)} \sum_{T \in \mathcal{T}_i} q_T \overset{\text{(a)}}{\leq} \frac{x(A_i)}{1 - x(A_i)}.$$

The last inequality (a) observes that the Galton-Watson produces at most one tree, hence all probabilities q_T sum to at most 1. □

Now we complete the proof of Theorem 17.2. For the expected length of the log, we now have

$$\mathbf{E}[L] \leq \sum_T \mathbf{P}[T_i = T \text{ for some } i] \leq \sum_T p(T) = \sum_{i=1}^n \sum_{T \in \mathcal{T}_i} p(T) \leq \sum_{i=1}^n \frac{x(A_i)}{1 - x(A_i)}.$$

To more precisely bound the expected number of times we resample an event A_i, we observe that A_i is resampled only if a tree $T \in \mathcal{T}_i$, with root labeled by A_i, is produced by the log. Thus

$$\mathbf{E}\left[\begin{matrix} \# \text{ times } A_i \\ \text{is resampled} \end{matrix}\right] \leq \sum_{T \in \mathcal{T}_i} \mathbf{P}[T_i = T \text{ for some } i] \leq \sum_{T \in \mathcal{T}_i} p(T) \leq \sum_{i=1}^n \frac{x(A_i)}{1 - x(A_i)},$$

as desired.

17.4 Additional notes and materials

We refer the reader to [MT10] for extensions including parallelization, a lopsided LLL condition, and derandomization. Additional applications of the local lemma can be found in [AS16]. We refer to [Tao09] for an alternative, information-theoretic perspective on Moser's [Mos09] original proof technique.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

17.5 Exercises

Exercise 17.1. A *hypergraph* is a generalization of an undirected graph where each edge may have more than 2 endpoints. A hypergraph has *uniform rank k* if every edge contains exactly k points. A hypergraph is *k-regular* if each vertex is incident to exactly k edges.

Design and analyze an algorithm that, for k sufficiently large[2], takes as input a k-regular hypergraph of uniform rank k and either: (a) outputs a vertex coloring such of 2 colors so that no edge is monochromatic, or (b) declares that no such coloring exists.

Exercise 17.2. Recall that a *proper vertex coloring* of an undirected graph G is one where no two adjacent vertices have the same coloring. Here we consider *vertex list-coloring*, where each vertex v is given a finite list L_v of colors, and we want a proper vertex coloring where each vertex is assigned a color from a list.

Design and analyze an algorithm that, given an undirected graph $G = (V, E)$ with maximum degree Δ, and lists of colors L_v for each vertex v of size $|L_v| \geq 10\Delta$, either computes a proper vertex list-coloring or declares that no such coloring exists.

[2]That is, the algorithm should work for for all $k \geq c$ for some universal constant c. $c = 9$ is possible.

Chapter 18

DNF counting

18.1 Unbiased estimators

Consider a unit disk D of radius 1, centered at 0. The area of D is π. Let $p \in [-1,1]^2$ be a uniformly random point in the unit square. We have

$$\mathbf{P}[p \in D] = \frac{\text{area of } D}{4} = \frac{\pi}{4}.$$

So we can try to estimate π by repeatedly sampling points in $[-1,1]^2$ and counting the number of samples lie in D. Let $p_1 \ldots, p_T \in [-1,1]^2$ be independently sampled uniformly at random. For each $t \in [T]$, let

$$X_t = \begin{cases} 1 & \text{if } p_t \in D \\ 0 & \text{otherwise.} \end{cases}$$

Consider the sum $\sum_{t=1}^T X_t/T$. We have $\mu \overset{\text{def}}{=} \mathbf{E}\left[\sum_{t=1}^T X_t\right] = T\pi/4$. The Chernoff bound states that

$$\mathbf{P}\left[\left|\frac{1}{T}\sum_{t=1}^T X_T - \frac{\pi}{4}\right| \geq \epsilon\right] \leq \mathbf{P}\left[\left|\sum_{t=1}^T X_T - \mu\right| \geq \epsilon\mu\right] \leq C_0 e^{-\mu\epsilon^2} \leq e^{-C_1 T\epsilon^2}$$

for universal constants $C_0, C_1 > 0$ (for sufficiently large T). In particular, $C = O(\log(n)/\epsilon^2)$ guarantees ϵ-error with high probability of success.

Of course this is just a standard application of concentration. However, to apply concentration, we needed the fact that $\pi/4$ was at most a constant. In general, the concentration argument only gives a tail inequality of the form

$$e^{-C\epsilon^2 T\,\mathbf{E}[X]},$$

for T independent trials of an unbiased and bounded estimator X, for some constant $C > 0$. We need $T \geq 1/C\epsilon^2\,\mathbf{E}[X]$ for concentration to kick in.

Above, $\mathbf{E}[X] = \pi/4 = .785...$, so $1/\mathbf{E}[X]$ was not significant. If $\mathbf{E}[X]$ were exponentially small, then we would need exponentially many trials. But there are also basic settings where the most natural unbiased estimator may be exponentially small.

18.2 DNF counting

Let φ be a DNF (*disjunctive normal form*) formula with n variables and m clauses; e.g.,

$$\varphi(x_1, \dots, x_n) = (x_1 \wedge \bar{x}_3 \wedge \cdots \wedge x_n) \vee (x_2 \wedge \cdots \wedge \bar{x}_{n-1})$$

It is easy to decide if a DNF formula is satisfiable. Consider instead the problem of *counting* the number of satisfying assignments to DNF.

The obvious unbiased estimator samples an assignment $x_1, \dots, x_n \in \{\texttt{true}, \texttt{false}\}$ uniformly at random, and tests if $\varphi(x_1, \dots, x_n) = \texttt{true}$. Then the number of satisfying assignments is 2^n times the probability that the formula is true. Alas, the probability that $\varphi(x_1, \dots, x_n) = \texttt{true}$ may be exponentially small, in which case exponentially many samples would be needed to get a reasonable approximation.

Let $\mathcal{A} = \{\texttt{true}, \texttt{false}\}^n$ be the family of all possible boolean assignments. For each clause $i \in [m]$, let $S_i \subseteq \mathcal{A}$ be the family of satisfying assignments for the ith clause. The total number of satisfying assignments is $|\bigcup_i S_i|$.

A more general view of our setup is as follows. We have n subsets $S_1, \dots, S_n$. For each i, we know $|S_i|$, and how to sample a point $x \in S_i$ uniformly at random. Our goal is to estimate the size of the union, $|S_1 \cup \cdots \cup S_n|$, but it is hard to do so directly. Can we use our knowledge about the individual S_i's to infer the size of the union?

Now consider the ratio

$$\mu = \frac{|\bigcup_i S_i|}{\sum_i |S_i|}.$$

Since we can calculate the denominator explicitly, an estimate μ gives an estimate for the numerator. Crucially, $\mu \geq 1/m$ because each satisfying assignment in the numerator appears in at most all m sets in the denominator. How can we estimate μ?

The denominator counts the satisfying assignments in the S_i's with repetition. We can think of this as the number of pairs (i, x) where $i \in [m]$ and $x \in S_i$. The numerator counts the satisfying assignments in the S_i's without repetition. This equals the number of pairs (i, x) where $i \in [m]$, $x \in S_i$, and i *is the first index such that* $x \in S_i$.

So we want to sample from the denominator, and see if it is in the numerator. This can be done as follows:

1. Pick a clause $i \in [m]$ in proportion to $|S_i|$, and an assignment $x \in S_i$ (satisfying the ith clause) uniformly at random.
2. If i is the first clause satisfied by x, return 1. Otherwise return 0.

This estimator can be implemented in polynomial time. Its expected value is μ. Since $\mu \geq 1/m$, the average of $O(m \log(n)/\epsilon^2)$ independent trials gives a $(1 \pm \epsilon)$-estimate of μ with high probability.

18.3 Additional notes and materials

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

18.4 Exercises

Exercise 18.1. Design and analyze a polynomial time subroutine implementing the unbiased estimator in Section 18.2. (The faster the better.)

Exercise 18.2. Let $f(x_1, \dots, x_n)$ be a DNF formula with m clauses and n random variables. Let $p_1, \dots, p_n \in [0, 1]$, and $\epsilon \in (0, 1)$. Consider the random assignment $x \in \{\texttt{false}, \texttt{true}\}^n$ where for each variable i independently, we have $\mathbf{P}[x_i = \texttt{true}] = p_i$.

Design and analyze a randomized algorithm that estimates the probability that $f(x) = \texttt{true}$ up to a $(1 \pm \epsilon)$-factor with high probability.[1]

[1]Hint: Define the *mass / probability measure* $\mu(S)$ of a set $S \subseteq \mathcal{A}$ by $\mu(S) \overset{\text{def}}{=} \mathbf{P}[x \in S] = \sum_{y \in S} \mathbf{P}[x = y]$. Consider the measure of each S_i, and the measure of the union $S_1 \cup \cdots \cup S_n$.

Chapter 19

A Linear Map of the Web

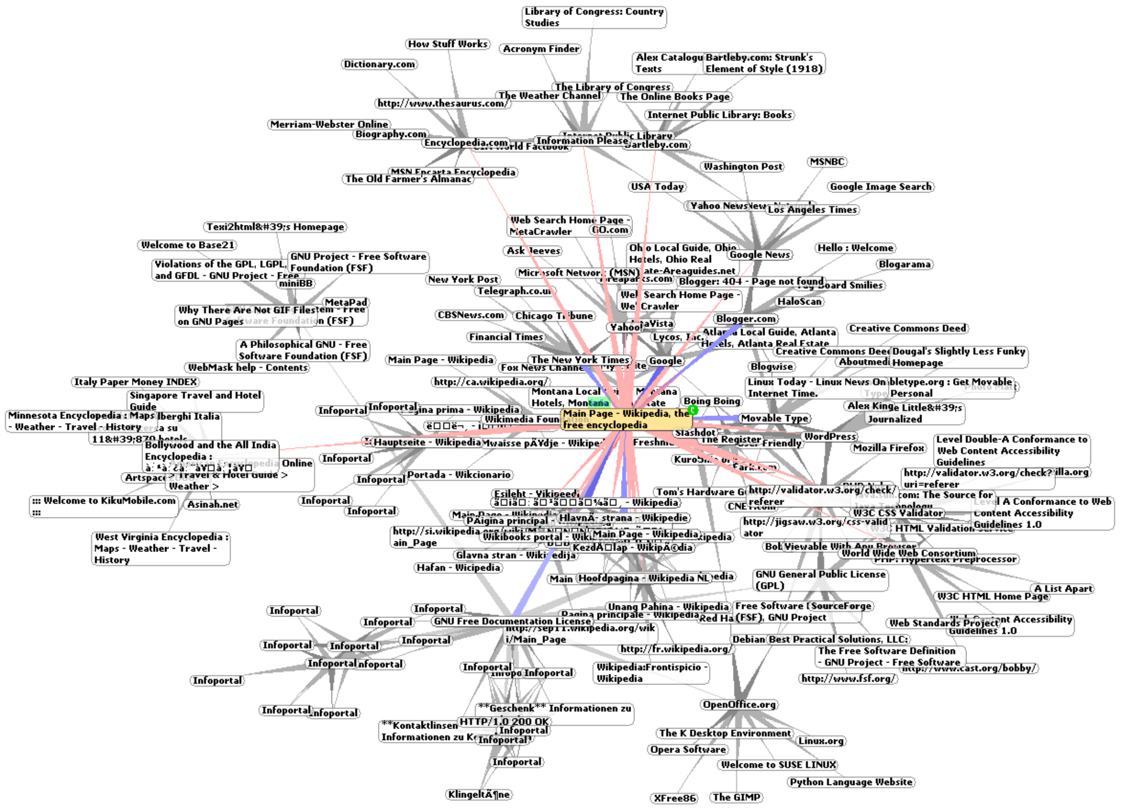


Figure 19.1: Map of a small part of the world wide web around `wikipedia.org` [CW].

19.1 Ranking the web

The world wide web is a large and messy place. As of March 2022, http://worldwidewebsize.com estimates that there are 2.97 *billion* indexed webpages. Even more amazing is that modern search engines, starting with Google, are able to process, organize and index this nearly unbounded corpus and make it useful. You can query for a topic of interest, and the search engine returns a long list of relevant websites, almost immediately. More often than not, you find what you are looking for within the first few listed results. This is utterly amazing, and we basically take it for granted.

It is one thing to identify all the web pages containing (or relevant to) a search query. This requires crawling the internet, and building a huge index that roughly identifies which keywords appear where. Some of the randomized data structures discussed earlier may be helpful for managing this task. But even within a search query, there seems to be an unlimited number of pages about a given topic, and a lot of it is junk. There is still another challenge to identify the best pages for the query. How do we separate the good websites from the bad? We should keep in mind the *scale* of the world wide web. It is pointless to try to evaluate the websites individually. This is a large scale *ranking* problem.

Modern search engines are based on the idea that the *link structure* of the world wide web reveals some sense of importance among the websites. When we write a paper, we cite the references that support or inform our argument. Likewise, web sites link to other websites and thereby implicitly bestow some approval. Another appeal of analyzing links is that we can model everything in basic graph theory, where we have good algorithms and sound analysis. The link structure gives a good starting point for our first idea for ranking webpages.

Idea 1. *Score each website equal to the number of other webpages linking to it.*

$$\text{score}_1(v) = \sum_{(u,v)\in E} 1.$$

One feature of score_1 is that every link out of a vertex u is worth 1 point. But if u has many outgoing links, shouldn't that dilute the "approval" bestowed by u? As an analogy, suppose we have two lists of movies. One lists the top 10 movies of all time, and the other lists the top 100 movies of all time. Shouldn't it be worth more to be on the first list? Our next proposal for ranking scales down the value of a link (u, v) by the number of outgoing edges of u, so that the total sum of links leaving u is 1.

Let $d^+(u)$ denote the number of edges leaving a vertex u, a.k.a. the *out-degree*.

Idea 2. *Score each website v as the sum, over all other webpages u linking to v, of* 1 *divided by the number of outgoing links from u:*

$$\text{score}_2(v) = \sum_{u:(u,v)\in E} \frac{1}{d^+(u)}.$$

Unfortunately score$_2$ can be manipulated too. To drive up score$_2(v)$ for a website v, one can create many fake websites u with a link to v. Perhaps, when evaluating a link (u, v), we should consider whether u is much of an authority to begin with.

Idea 3. *Score each website v equal to the weighted sum, over all other webpages u linking to it, of the score of u divided by the number of outgoing links from u:*

$$\text{score}_3(v) = \sum_{u:(u,v)\in E} \frac{\text{score}_3(u)}{d^+(u)}.$$

score$_3(v)$ attains a sort of self-consistent nirvana. For example, the weight of a link (u, v), in attributing authority to v, is adjusted in proportion to the authority of u. The total authority distributed by a webpage u is exactly equal to u's own authority, score$_3(u)$.

While this recursive relationship is appealing, there is no reason, *a priori*, why such scores should exist.

Today's discussion is about how $\{\text{score}_3(v),\ v \in V\}$ *does* exist, and the structure and interpretations thereof. This happy miracle is entirely due to the fact that the values $\{\text{score}_3(v), v \in V\}$ satisfy a particularly well-structured *linear system of equations*. As such, our discussion will soon be translated into linear algebra. An important part of the structure comes from a probabilistic interpretation of the linear system.[1]

The goal of this discussion is to prove the following theorem (in more general terms). Recall that $\Delta^V = \left\{x \in \mathbb{R}^V_{\geq 0} : \sum_{v\in V} x_v = 1\right\}$ denotes the set of probability vectors over V.

Theorem 19.1. *There exists a vector $x \in \Delta^V$ such that*

$$x_v = \sum_{u:(u,v)\in E} \frac{x_u}{d^+(u)} \tag{19.1}$$

for all $v \in V$. If G is strongly connected, then this vector is unique and strictly positive.

[1]Naturally, the mathematics we discuss existed long before search engines and similar ideas had been applied elsewhere.

To try to understand where such an x comes from — and in particular, how we can assert that it describes a distribution over B — consider the following *random walk* on G. At each step, you are on some vertex u. You choose an outgoing edge $(u, v) \in E$ uniformly at random, and step to v. This may send you walking chaotically all over the graph. On the world wide web, this is like randomly surfing the web where you keep following randomly chosen links. If the links tend to point to useful websites, then over time your random walk should stumble upon good web sites more often then a uniformly random sample of the web. Now, depending on where you start, after some number of k steps, there are different probabilities of where you would end up. As k increases, we might hope this reaches some kind of equilibrium, where the distribution is the same or close to the same between to kth and $(k+1)$th step for sufficiently large k. This brings us to the notion of a stationary distribution.

Definition 19.2. *Fix a random walk on a set of vertices V. A set of probabilities $x \in \Delta^V$ is a* stationary distribution *for the random walk if taking a random step from the distribution x produces the same distribution x.*

Consider again Theorem 19.1, and consider the recursive relations satisfied by $x \in \Delta^V$:

$$x_v = \sum_{u:(u,v)\in E} \frac{x_u}{d^+(u)}$$

for all $v \in V$. In our random walk, from a given vertex u, we choose an outgoing edge (u, v) with probability $1/d^+(u)$. In particular, if u is chosen from a random distribution $x \in \Delta^V$, then the probability of then stepping to a particular vertex v is

$$\sum_{u:(u,v)\in E} \frac{x_u}{d^+(u)}.$$

For the vector x asserted by Theorem 19.1, this sum equals x_v. That is, *x is the stationary distribution of a random walk on G.* We can restate Theorem 19.1 as follows.

Theorem 19.1, restated. *Every random walk on a directed graph G has a stationary distribution. If G is strongly connected, then this distribution is unique.*

19.1.1 PageRank

The actual PageRank algorithm proposed in [PBM+99] is slightly different. We augment the random walk thought experiment described above with a small probability

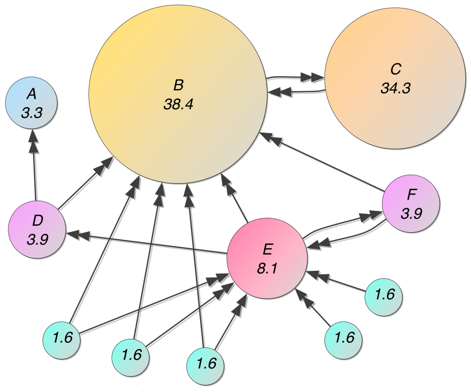


Figure 19.2: An example of PageRank on an 11 vertex graph [Wika].

α of restarting the walk from a page chosen uniformly at random. Otherwise (with the remaining probability $1-\alpha$), we continue the random walk as described above. The same conclusion holds for this random walk.

Theorem 19.3. *Let $G=(V,E)$ be a directed graph, and let $\alpha \in (0,1)$. There exists a unique vector $x \in \Delta^V$ such that for all $v \in V$,*

$$x_v = (1-\alpha) \sum_{u:(u,v)\in E} \frac{x_u}{d^+(u)} + \frac{\alpha}{n}.$$

The PageRank formulation has some additional convenient properties compared to general random walks. First, the vector x is guaranteed to be unique, and has strictly positive coordinates. (Implicitly, it is the stationary distribution on the directed graph augmented by weighted edges between all pairs, which by Theorem 19.1 is unique.) Second, it can be calculated more directly. We will come back to this point at the end of our discussion in Section 19.6.

19.2 A linear map of the web

Recall that a function $f : \mathbb{R}^n \to \mathbb{R}^n$ is *linear* if it satisfies the following:

$$f(x+y) = f(x) + f(y)$$

Let $A : \mathbb{R}^V \to \mathbb{R}^V$ be the linear map encoding the directed edges as follows. For a vertex v, let $e_v \in \{0,1\}^V$ be the vector with 0's everywhere except for 1 in the vth coordinate. We define A by setting $(Ae_v) \in \{0,1\}^V$ to indicate the outgoing neighbors

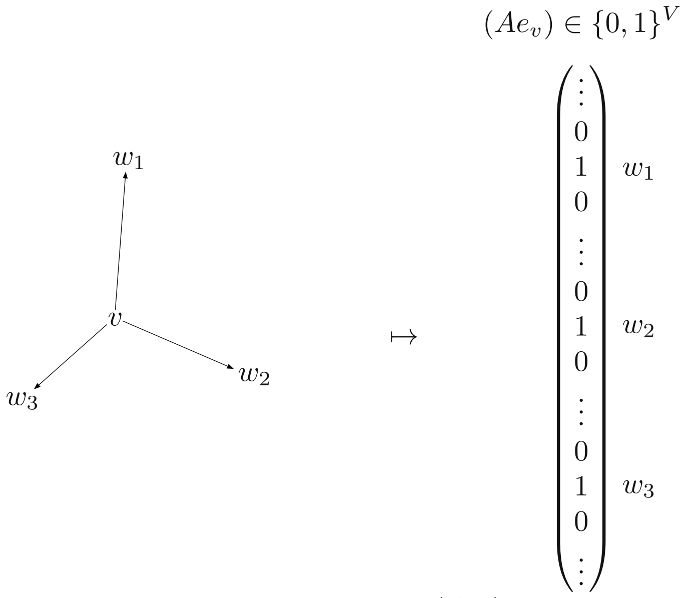


Figure 19.3: The adjacency vector (Ae_v) of a vertex v.

of v. More explicitly, A is defined by

$$\langle e_w, Ae_v \rangle = (Ae_v)_w = \begin{cases} 1 & \text{if } (v,w) \in E \\ 0 & \text{otherwise.} \end{cases}$$

See also Fig. 19.3. Note that the graph G is undirected iff A is symmetric.

More generally, we might have weights on the edges of the graph. Then we would define $A : \mathbb{R}^V \to \mathbb{R}^V$ by

$$\langle e_w, Ae_v \rangle = (A^T e_v)_w = \begin{cases} \text{weight of the edge } (v,w) & \text{if } (v,w) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Let us rehash our discussion on scoring websites in terms of the linear map. We have

$$\text{score}_1(v) = \langle e_v, A\mathbb{1} \rangle = \sum_w (Ae_v)_w.$$

We also have

$$d^+(u) = \langle \mathbb{1}, Ae_u \rangle.$$

Let $D = \operatorname{diag}(d^+) \in \mathbb{R}^{V \times V}$ be the diagonal matrix induced by d^+. That is, for any vector x and vertex v, we have

$$(Dx)_v = d^+(v) x_v.$$

Then we can write $\operatorname{score}_2(v)$ as

$$\operatorname{score}_2(v) = \sum_{u:(u,v)\in E} \frac{1}{d^+(u)} = (AD^{-1}\mathbb{1})_v = \left\langle e_v, AD^{-1}\mathbb{1} \right\rangle.$$

As for the stationary distribution (score_3), we see that the stationary distribution x must satisfy the equation

$$x = Rx \text{ for the linear map } R = AD^{-1} : \mathbb{R}^V \to \mathbb{R}^V.$$

The map R has the property that $Rx \in \Delta^V$ for any $x \in \Delta^V$. Any linear map $A : \mathbb{R}^n \to \mathbb{R}^n$ mapping Δ^n to Δ^n is called a *stochastic* linear function.

19.3 Eigenvectors

Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. An *eigenvector* of A is a nonzero vector $x \neq \mathbb{0}$ such that

$$Ax = \lambda x$$

for some scalar value $\lambda \in \mathbb{C}$. The scalar λ is called an *eigenvalue* of A, and here it is the eigenvalue corresponding to the eigenvector x.

Alternatively, a value $\lambda \in \mathbb{C}$ is an eigenvalue iff the linear map $(A - \lambda I) : \mathbb{R}^n \to \mathbb{R}^n$ is not invertible. Indeed, any eigenvector x corresponding to λ gives a second vector (besides $\mathbb{0}$) such that $(A - \lambda I)x = \mathbb{0}$.

Lemma 19.4. *The set of eigenvectors corresponding to an eigenvalue λ form a vector space.*

The dimension of the subspace corresponding to an eigenvalue λ is called the *multiplicity* of the eigenvalue. For any eigenvalue λ of $A : \mathbb{R}^n \to \mathbb{R}^n$, the multiplicity of λ is equal to $n - \operatorname{rank}(A - \lambda I)$.

Existence of eigenvectors. *A priori*, it is not clear why every matrix should have an eigenvector.

Lemma 19.5. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. Then A has an eigenvalue $\lambda \in \mathbb{C}$ and an eigenvector $x \in \mathbb{C}^n$.*

Proof. Fix any nonzero vector v. Recall that any set of $n+1$ vectors in $\mathbb{R}^n$ is linearly dependent. In particular, the set $v, Av, \dots, A^n v$ is linearly dependent. Put alternatively, there is a nonzero, degree $k \leq n$ polynomial $p(x) = \alpha_k x^k + \cdots + \alpha_0$ such that

$$p(A)v = 0.$$

By the fundamental theorem of algebra, the polynomial $p(x)$ can be expressed as

$$p(x) = (x - r_1)(x - r_2)\cdots(x - r_k)$$

where $r_1, \dots, r_n \in \mathbb{C}$ are complex roots of $p(x)$. Then

$$p(A)v = (A - r_1 I)(A - r_2 I)\cdots(A - r_k I)v = 0$$

imples that some $A - r_i I$ maps a nonzero vector to $\mathbb{0}$. The corresponding root r_i is an eigenvalue. □

Tranposing and Eigenvalues The eigenvalues and eigenvectors of a matrix A and its transpose A^T are closely related. This is primarily because A and its transpose A^T have the same rank, and eigenvalues are ultimately concerned with values λ for which $A - \lambda I$ is not full rank.

Lemma 19.6. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. Then A is invertible iff A^T is invertible.*

Proof. Suppose A is invertible. We claim that A^T is invertible with inverse $(A^{-1})^T$. Indeed, for any two points x, y, we have

$$\left\langle x, (A^{-1})^T A^T y \right\rangle = \left\langle A^{-1} x, Ay \right\rangle = \left\langle AA^{-1} x, y \right\rangle = \langle x, y \rangle.$$

Since this holds for all x and y, we have that $(A^{-1})^T A = I$. That is, $(A^{-1})^T$ is an inverse for A. This proves the "only if" whereas we claim "if and only if"; the "if" follows symmetrically as $(A^T)^T = A$. □

Lemma 19.7. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map.. Then A and A^T have the same eigenvalues with the same multiplicities.*

Proof. Recall that for any map $L : \mathbb{R}^n \to \mathbb{R}^n$, L is invertible iff L^T is invertible, and $(L^T)^{-1} = L^T$. Now suppose that λ is an eigenvalue of A. Then $A - \lambda I$ is not invertible, hence $(A - \lambda I)^T = A^T - \lambda I$ is not invertible, and so λ is an eigenvalue of A. The multiplicities are equal because the

$$\operatorname{rank}(A - \lambda I) = \operatorname{rank}\Big((A - \lambda I)^T\Big) = \operatorname{rank}(A - \lambda I).$$

□

Lemma 19.8. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map.. Let x be an eigenvector of A and let y be an eigenvector of A^T corresponding to distinct eigenvalues. The $\langle x, y \rangle = \mathbb{0}$.*

Proof. Let x have corresponding eigenvalue λ and let y have corresponding eigenvalue μ. We have

$$\lambda\langle x, y\rangle = \langle Ax, y\rangle = \Big\langle x, A^T y\Big\rangle = \mu\langle x, y\rangle.$$

If $\lambda \neq \mu$, then $\langle x, y\rangle = 0$. □

Let us now restate Theorem 19.1 in our new language of eigenvectors and eigenvalues.

Theorem 19.1, in terms of eigenvectors and eigenvalues. *Let $G = (V, E)$ be a directed graph and let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the map corresponding to the linear map corresponding to the random walk on G. Then R has an eigenvector $x \in \Delta^V$ with eigenvalue 1. If G is strongly connected, then x is the unique eigenvector (up to scaling) with eigenvalue 1.*

19.4 The Perron-Frobenius theorem

Definition 19.9. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. A is* positive *if $\langle x, Ay\rangle > 0$ for all $x, y \in \mathbb{R}^n_{\geq 0}$ with $x, y \neq \mathbb{0}$.*

Equivalently, $Ax > \mathbb{0}$ (coordinatewise) for all nonzero $x \in \mathbb{R}^n_{\geq 0}$. The following theorem is called the *Perron-Frobenius theorem* and shows that positive linear maps have a lot of structure.

Theorem 19.10. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a positive linear map. Then A has an eigenvalue λ_1 with eigenvector x_1 with the following properties.*

1. *$\lambda_1 > 0$ and $x_1 > \mathbb{0}$.*

2. *x_1 is the unique (generalized) eigenvector of λ_1.*

3. *x_1 is also the unique nonnegative vector such that $Ax_1 \geq \lambda_1 x_1$ (up to scaling).*

4. *Any other eigenvalue μ has $|\mu| < \lambda_1$.*[2]

5. *Any other eigenvector of A has at least one negative entry.*

Proof. Let

$$L = \{\lambda > 0 \text{ where } Ax \geq \lambda x \text{ for some } x \in \Delta^n\}.$$

We will argue that the the supremum of L is the desired value λ_1. To this end, we first make the following claims about L.

1. *L is nonempty.*

2. *L is bounded.*

3. *L is closed.*

For claim 1, let $x \in \Delta^n$ with $x > \mathbb{0}$. The Ax will have strictly positive coordinates because A is positive, and so there is some $\lambda > 0$ such that $Ax > \lambda x$.

For claim 2, we observe that for $\lambda \in L$, with (say) $Ax \geq \lambda x$ with $x \in \Delta_n$, we have

$$\langle \mathbb{1}, A\mathbb{1} \rangle \overset{\text{(a)}}{\geq} \langle \mathbb{1}, Ax \rangle \overset{\text{(b)}}{\geq} \langle \mathbb{1}, \lambda x \rangle \overset{\text{(c)}}{=} \lambda.$$

(a) is because A is monotonic and $\mathbb{1} \geq x$. (b) is by choice of x and λ. (c) is because $x \in \Delta_n$.

For claim 3, let $\lambda_1, \lambda_2, \ldots$ be any sequence of points in L that converges to some $\bar{\lambda}$. We want to show that $\bar{\lambda} \in L$. For each i, let $x_i \in \Delta^n$ with $Ax_i \geq \lambda_i x_i$. Since Δ^n is compact, a subsequence $x_{i_1}, x_{i_2}, \ldots$ of the x_i's converge to some $\bar{x} \in \Delta^n$. We have

$$A\bar{x} = A\left(\lim_{j \to \infty} x_{i_j}\right) \overset{\text{(d)}}{=} \lim_{k \to \infty} Ax_{i_j} \geq \lim_{j \to \infty} \lambda_{i_j} x_{i_j} = \lambda \bar{x}.$$

(d) invokes continuity of A to pass through the limit.

We have now shown that L is a closed and bounded set with at least one positive number. Any nonempty closed and bounded set L *has a finite supremum, λ_1, which is contained in L.* By definition of L, there is also a vector $x_1 \in \Delta^n$ such that $Ax_1 \geq \lambda_1 x_1$. *We claim that $Ax_1 = \lambda_1 x_1$.*

[2]We will only prove that $|\mu| \leq \lambda_1$.

Let $y \geq \mathbb{0}$ be such that $Ax_1 = \lambda_1(x_1 + y)$. Then $Ax_1 = \lambda_1 x_1 \iff y = \mathbb{0}$. Suppose by contradiction that $y \neq \mathbb{0}$. Choose $\epsilon \in (0,1)$ sufficiently small that

$$\frac{\epsilon}{1-\epsilon} \leq \langle \mathbb{1}, y \rangle.$$

Observe that

$$(1-\epsilon)x_1 + \frac{\epsilon}{\langle \mathbb{1}, y \rangle} \in \Delta^n.$$

Moreover, we have

$$A\left((1-\epsilon)x_1 + \frac{\epsilon}{\langle \mathbb{1}, y \rangle} y\right) \overset{\text{(e)}}{>} (1-\epsilon)Ax_1 \overset{\text{(f)}}{=} (1-\epsilon)\lambda(x_1 + y) \overset{\text{(g)}}{\geq} \lambda\left((1-\epsilon)x_1 + \frac{\epsilon}{\langle \mathbb{1}, y \rangle} y\right).$$

Here (e) is because $Ax > \mathbb{0}$ for all $x \geq \mathbb{0}$ with $x \neq \mathbb{0}$. (f) is by choice of y. (g) is by choice of ϵ. The strict inequality obtained above implies that there is a larger value than λ in L, a contradiction. Thus we must have $Ax_1 = A\lambda_1$ after all.

Note also that x_1 is strictly positive, as $x_1 = \lambda^{-1}Ax_1 > \mathbb{0}$. More generally, any eigenvector of A associated with a positive eigenvalue is strictly positive.

Next we claim that x_1 is the unique (simple) eigenvector for λ_1 (up to scaling). Indeed, suppose $Ay = \lambda_1 y$ for vector y. As mentioned above, we must have $y > \mathbb{0}$. If y is not proportional to x, then let $z = x - \alpha y$ where $\alpha > 0$ is such that $z \geq 0$, $z \neq \mathbb{0}$, and $z_i = 0$ for some coordinate i. Then z would be an eigenvector of A that is not strictly positive, a contradiction.

Now we claim that x_1 is the unique generalized eigenvector for λ_1, as well. If not, then there would a vector y not spanned by x_1 such that

$$(A - \lambda_1 I)^2 y = 0.$$

Then $(A - \lambda_1 I)y$ is a simple eigenvector of A with eigenvalue λ_1, so

$$Ay = \lambda_1 y + cx_1 \tag{19.2}$$

for some $c \neq 0$. By flipping the sign of y if necessary, we may assume that $c > 0$. Increase y by a multiple of x_1 if necessary, we may assume that $y > 0$. If $y > 0$ and $c > 0$, then (2) that there is a value larger than λ_1 in L, a contradiction. Thus x_1 is the unique generalized eigenvector for λ_1.

Take any other eigenvalue μ of A, with eigenvector y. Scale y such that Then

$$|\mu||y| = |\mu y| = |Ay| \leq A|y|,$$

where $|y|$ denotes the coordinate-wise absolute value of y. Thus $|\mu| \in L$, and therefore $|\mu| \leq x_1$.

If $|\mu| = \lambda$, then we would have $|y| = x_1$ (after scaling) by uniqueness of x_1. This, combined with $|Ay| = A|y|$, implies (by known facts about complex numbers, which we omit) that $y = e^{i\theta} x_1$ for some fixed θ. Thus $y \in \operatorname{span}(x_1)$ and $\mu = x_1$.

For the last claim, observe that A^T also satisfies the hypothesis. Indeed, if $x, y \geq \mathbb{0}$ and neither x nor y equals zero, then

$$\left\langle A^T x, y \right\rangle = \langle x, Ay \rangle > 0.$$

Thus A^T has (the same) dominant eigenvalue λ_1, with a positive eigenvector $y_1 \in \mathbb{R}^n_{>0}$. Now, any eigenvector x of A corresponding to an eigenvalue other than x_1 must be orthogonal to y_1. If y_1 is has strictly positive coordinates and $\langle x, y_1 \rangle = 0$, then x must have at least one negative coordinate. $\square$

The above proof is essentially due to Bohnenblust; see [Lax07; Bel97].

19.5 Perron-Frobenius for strongly connected random walks

We now extend Theorem 19.10 to random walks. In particular, we will show that there is always a stationary distribution, and that this stationary distribution is unique if the underlying graph is strongly connected.

Theorem 19.11. *Let $A : \mathbb{R}^V \to \mathbb{R}^V$ be the linear map of a random walk on a strongly connected graph.*

1. *There is an eigenvector $x \in \Delta^n$ with eigenvalue 1 and $x > \mathbb{0}$.*
2. *x is the unique eigenvector with eigenvalue 1.*
3. *x is the only eigenvector of A with no negative entries.*
4. *Any other eigenvalue μ has $|\mu| \leq 1$.*

Proof. We first note that A has eigenvalue 1. Indeed, because A models a random walk, we have

$$A^T \mathbb{1} = \mathbb{1},$$

as can be verified directly. Thus 1 is an eigenvalue of A^T and thereby an eigenvalue of A as well. Now, let x be any eigenvector of A, rescaled so that $\langle \mathbb{1}, x \rangle = 1$. We claim that $x \in \Delta^n$.

Consider the matrix $B = \frac{1}{n} \sum_{i=0}^{n} A^i$. We can interpret B as the random walk on V induced by the following two steps:

1. Choose $i \in \{0, \dots, n-1\}$ uniformly at random.
2. Take i steps of the random walk A.

Observe that any eigenvalue μ of A corresponds to an eigenvalue of B with value

$$\frac{1}{n}\sum_{i=0}^{n-1} \mu^i,$$

with the same eigenvectors. In particular, 1 is an eigenvector of x with eigenvalue 1.

Because G is strongly connected, there is a path of at most $n-1$ edges from any point $a \in V$ to any point $b \in V$. Thus B models a random walk with strictly positive transition probabilities for all pairs of vertices. In particular, B satisfies the positive assumptions of Theorem 19.10.

Let $\lambda_1 > 0$ and $x_1 \in \Delta^n$ be the "dominant" eigenvector and eigenvalue. We claim that $\lambda_1 = 1$ and $x_1 = x$. To this end, consider B^T. B^T has the same dominant eigenvalue λ_1, with a corresponding eigenvector that is also the only eigenvector with no negative corodinates. We also know that 1 is an eigenvalue of B^T with eigenvector $\mathbb{1}$. So $\lambda_1 = 1$. Theorem 19.10 then implies that x is the unique eigenvector for eigenvalue 1, and all the coordinates of x are strictly positive.

Let μ be any other eigenvalue of A. We claim that $|\mu| \leq 1$. For $\epsilon \geq 0$, let $A_\epsilon = (1-\epsilon)A + \epsilon B$. Let

$$\mu_\epsilon = \mu + \frac{\epsilon}{n}\sum_{i=1}^{n-1} \mu^i.$$

μ_ϵ is an eigenvalue of A_ϵ for all $\epsilon \geq 0$. Note that for all $\epsilon > 0$, Theorem 19.10 applies to A_ϵ and in particular A_ϵ has dominant eigenvalue 1, and $|\mu_\epsilon| \leq 1$. Moreover, $\lim_{\epsilon \to 0} \mu_\epsilon = \mu$. Thus $|\mu| \leq 1$.

Any eigenvector other than x is an non-dominant eigenvector of B, and thus has negative coordinates. □

19.6 Computing PageRank

The PageRank vector x satisfies the equation

$$x = (1-\epsilon)Rx + \frac{\epsilon}{n}\mathbb{1},$$

where $R = AD^{-1}$ models the random walk on the directed graph, and where $\mathbb{1} \in \mathbb{R}^V$ is the all-1's vector. We can rewrite this as

$$\left(\left(\frac{1}{1-\epsilon}\right)I - R\right)x = \frac{\epsilon}{(1-\epsilon)n}\mathbb{1}.$$

Recall that the maximum eigenvalue of R is 1; in paricular, $1/(1-\epsilon)$ is not an eigenvalue. Thus $(1/(1-\epsilon)I - R)$ is invertible. We have

$$x = \left(\frac{\epsilon}{(1-\epsilon)n}\right)\left(\frac{1}{1-\epsilon}I - R\right)^{-1}\mathbb{1} = \frac{\epsilon}{n}(I - (1-\epsilon)R)^{-1}\mathbb{1}.$$

We can write $(I - (1-\epsilon)R)^{-1}$ as the infinite series

$$(I - (1-\epsilon)R)^{-1} = \lim_{k\to\infty}\sum_{i=0}^{k}((1-\epsilon)R)^{i}.$$

Thus

$$x = \frac{\epsilon}{(1-\epsilon)n}\lim_{k\to\infty}\sum_{i=0}^{k}((1-\epsilon)R)^{i}\mathbb{1}.$$

The series on the RHS converges quickly for moderate ϵ, so in practice one only has to compute a few terms in the sum.

19.7 Additional notes and materials

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

19.8 Exercises

Exercise 19.1. Let $G = (V, E)$ be a directed graph, not necessarily strongly connected. Recall that the strongly connected componentsm for a directed acyclic graph. A *sink* *component* is a strongly connected component with no out going edges; i.e., a sink in the DAG of strongly connected components. Suppose G has a unique sink component $S \subset V$. Show that the random walk on G has a unique stationary distribution $x \in \Delta^V$ and that $x_v > 0$ iff $v \in S$.

Exercise 19.2. Let $G = (V, E)$ be a simple[3], unweighted, directed, and strongly connected graph. We proved in Theorem 19.1 that the random walk G has a unique and strictly positive stationary distribution $x \in \Delta^V$. Prove that for all $v \in V$, $x_v \geq n^{-(n+1)}$.

[3] *Simple* means that there are no parallel edges.

Chapter 20

Connectivity and Electricity

20.1 Introduction

Connectivity is one of the simplest graph problems and one that is discussed in every introductory algorithms class. Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Two vertices $s, t \in V$ are *connected* if there is a path between them. A simple graph problem is to decide if two vertices are connected in a graph. Sometimes we also ask for the path between them.

The problems can be solved by simple *search algorithms*. The most common ones are *breadth first search* and *depth first search*. Both of these algorithms *mark* the vertices that they visit to avoid revisiting vertices. Like Hansel and Gretel dropping bread crumbs.

Complexity theorists are interested not only in the running time required by a problem, but also the amount of space required. This is the amount of space *in addition* to the input. A natural lower bound for any (nontrivial) problem is *logarithmic* in the input size, since we need $\lceil \log n \rceil$ bits just to represent the location of a pointer amongst an input of size n. Complexity theorists ask: what problems can solved in $O(\log n)$ space?

Consider connectivity. Search algorithms like BFS or DFS need $O(n)$ space to mark the vertices. Can (s, t)-connectivity by solved in *less* than $O(n)$ space? Surprisingly, the answer is yes. Savitch's theorem gives a $O\left(\log^2(n)\right)$ space algorithm, though it is not polynomial time [Sav70]. Now, can (s, t)-connectivity by solved in $O(\log n)$ space? This is basically just enough space to keep track of what vertex you are currently on as you search a graph.

Consider the following simple algorithm. Given $s, t \in V$, start a random walk from s, where in each iteration you pick a random neighbor of the current vertex and move there. Randomly walk for $O(mn)$ steps. If you come across t at any point, then you know that s and t are connected. If not, then you answer no.

Clearly the above algorithm takes only $O(\log n)$ space. You only need to keep track of which vertex you are on, and generate a random number between 1 and (at most) n to randomly select a neighbor. You also need to count the number of steps you've taken so far. When the algorithm answers in the affirmative, you are always right. Unfortunately, if it answers no, the algorithm could be wrong. Certainly there is no proof that there is no path from s to t. But today we will show that the algorithm is correct with constant probability. Therefore you can rerun the experiment $O(\log n)$ times to be correct with high probability.

To frame the analysis we introduce the following definition.

Definition 20.1. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Consider a random walk starting from a vertex s. The* hitting time *from s to a vertex v, denoted $H(s, v)$ is the expected number of steps until a random walk from s reaches v. The* cover time *from s, denoted $C(s)$, is the expected number of steps until a random walk from s visits every vertex in the graph.*

The above quantities may be infinite when the graph is not connected. We can rephrase the question of (s, t)-connectivity as asking if $H(s, t)$ is finite. Today we will prove the following.

Theorem 20.2. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $s \in V$. Then the cover time $C(s)$ is bounded above by $C(s) \leq m(n - 1)$.*

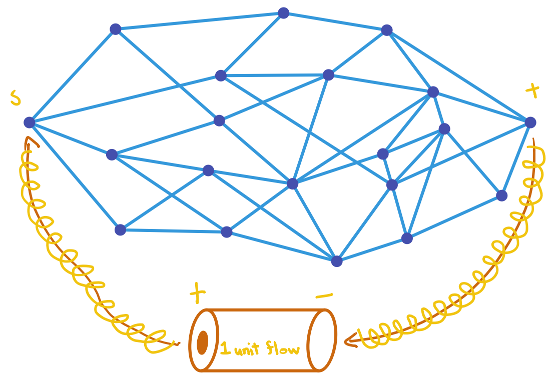
Assuming for the moment that the above theorems are true, and that random walks can decide (with high probability) whether vertices are connected, there is one big question left to answer:

Can (s, t)-connectivity be decided deterministically *in $O(\log n)$ space?*

We will return to this question later in Chapter 23.

20.2 Electrical networks

Our analysis will be based on a (perhaps surprising) connection between random walks and *electrical networks*. For the sake of our discussion, an *electrical network* is an undirected graph $G = (V, E)$ with positive edge weights $r : E \to \mathbb{R}_{>0}$ called *resistances*. If one attaches a battery to two vertices s and t, it induces a current that (in our discussion) is a unit flow from s to t. As the electricity flows from s to t, it is said to take the *path of least resistance*. What is the path of least resistance? A computer scientist or operations researcher might suggest this is the shortest path from s to t with respect to resistance. But physics does not do combinatorial optimization;

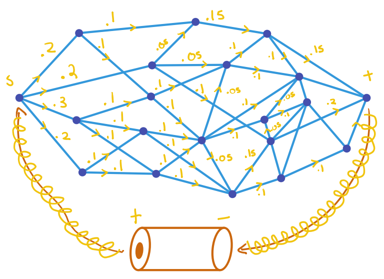


physics does calculus. From a calculus point of view, it is more natural to minimize a *sum of squares*.

To formalize the model, fix an orientation on the edges. We identify each flow with a vector $f \in \mathbb{R}^E$. For each edge e, a positive value $f(e) > 0$ means that $f(e)$ units of flow are routed in the same direction as the orientation of e. A negative value $f(e) < 0$ means that $|f(e)|$ units of flow are routed in the opposite direction. Then the electrical flow is the

$$(s,t)\textit{-flow } f \textit{ minimizing the } \text{electric energy } \sum_e r_e f_e^2 .$$

That is, the electrical flow is the solution of a constrained optimization problem. The squared terms encourage the flow to spread out. While there is still some preference to shorter paths, rather than put all the flow along the shortest (s,t)-path, the electrical flow will spread out such as in the example on the right. (The flow on the right is not actually the optimum electrical flow.)



Let us define a linear map $B : \mathbb{R}^E \to \mathbb{R}^V$ that maps flows to the net flows at each vertex. That is, for a flow f, and a vertex v, we have

$$(Bf)_v = \text{net flow of } f \text{ at } v.$$

The above definition is linear in f, so B is a linear map. Let $d \in \mathbb{R}^V$ be the demands of our problem; namely,

$$d_v = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \text{otherwise.} \end{cases}$$

A flow f is a unit (s,t)-flow iff $Bf = d$. Now, while we are principally interested in (s,t)-flow, the following discussion extends to any set of flow demands $d \in \mathbb{R}^V$ (the only requirement being that $\langle \mathbb{1}, d \rangle = 0$). In the above algebraic notation, the electrical flow is obtained as the solution to the following optimization problem:

$$\text{minimize } \langle f, Rf \rangle = \sum_{e \in E} r_e f_e^2 \text{ over } f \in \mathbb{R}^E \text{ s.t. } Bf = d. \tag{20.1}$$

For a fixed electrical network, the quantity above is a function of d. In general, for $d \in \mathbb{R}^V$ with $\langle \mathbb{1}, d \rangle = 0$, the *effective resistance of* d is the minimizing potential obtained by the electrical flow routing d.

The rest of this discussion is broadly organized into two parts.

1. The first part is about understanding the structure of an electrical flow. This is based on studying the first-order optimality conditions of (20.1).
2. The second part is about interpreting hitting times and cover times via electrical networks, and proving the desired bounds.

20.3 Structure of electrical flows

We have seen the electrical flow minimizes a sum of squares subject to linear constraints. This already endows a lot of structure to electrical flows by understanding the optimality conditions of such a problem. The connection to graphs then leads to further interpretations of these conditions.

20.3.1 Convex optimization s.t. linear constraints

The reader may recall that for *unconstrained* convex minimization problems, a point x is a minimum solution iff the derivative of the objective function is 0. This is no necessarily true in constrained optimization. For linear constraints, however, we know the following.

Theorem 20.3. *Consider a minimization problem of the form*

$$\text{minimize } \varphi(x) \text{ over } x \in \mathbb{R}^m \text{ s.t. } Ax = b,$$

where $\varphi : X \to \mathbb{R}$ is a convex and smoothly differentiable function over a vector space X, $A : X \to Y$ is a linear map, and $b \in Y$ is a vector. Let x be a optimum solution to the problem. Then

$$\varphi'(x) = A^T y \text{ for some } y \in \mathbb{R}^n.$$

Proof. Let $\ker(A) = \{x : Ax = \mathbb{0}\}$ denote the *kernel* of A; i.e., the set of vectors that map to $\mathbb{0}$.

Claim. $\varphi'(x)$ is orthogonal to $\ker(A)$. Suppose not. Then there exists $z \in \ker(A)$ such that

$$\langle \varphi'(x), z \rangle < 0.$$

But then for sufficiently small $t > 0$,

$$\varphi'(x + tz) \approx \varphi(x) + t\langle \varphi'(x), z \rangle < \varphi(x),$$

while

$$A(x + tz) = Ax + tAz = Ax = b.$$

Then $x + tz$ is not optimal, a contradiction, and proving the claim.

Now, recall that the *image of* A, denoted by $\operatorname{im}(A)$, is the set

$$\operatorname{im}(A) \overset{\text{def}}{=} \{Ax : x \in X\},$$

and that the *coimage of* A, denoted $\operatorname{coim}(A)$, is the subspace of X orthogonal to $\ker(A)$:

$$\operatorname{coim}(A) \overset{\text{def}}{=} X/\ker(A) = \{x \in X : \langle x, y\rangle = 0 \text{ for all } y \in \ker(A)\}.$$

Note that we have shown that $\varphi'(x)$ is orthogonal to A. Basic linear algebra (sometimes called the "fundamental theorem of linear algebra") states that A and A^T both induce isomorphisms (i.e., one-to-one linear mappings) between $\operatorname{coim}(A)$ and $\operatorname{im}(A)$. The one-to-one mapping $A^T : \operatorname{im}(A) \to \operatorname{coim}(A)$ implies that there exists $y \in Y$ such that $A^T y = \varphi'(x)$. ☐

20.3.2 Ohm's Law

If we apply Theorem 20.3 to the electrical flow problem, then we obtain the following identity called *Ohm's law*. The vector $p \in \mathbb{R}^V$ in the following theorem is called the *electric potentials* induced by d.

Theorem 20.4. *A flow $f \in \mathbb{R}^E$ subject to demands d is the electrical flow iff there exists $p \in \mathbb{R}^V$ such that $f = R^{-1}B^T p$.*

Proof. Suppose f is the electrical flow. Observe that the gradient of the objective function is Rf. By Theorem 20.3, there exists $q \in \mathbb{R}^V$ such that $2Rf = Bq$; hence $p = q/2$ is the desired set of electrical potentials.

Conversely, suppose $f = R^{-1}B^T p$ for some $p \in \mathbb{R}^V$. Let g be any other flow with $Bg = d$. Recall that for any convex function φ, we have

$$\varphi(y) \geq \varphi(x) + \langle \varphi'(x), y - x\rangle.$$

For our convex function $\varphi(f) = \langle f, Rf\rangle/2$, we have

$$\begin{aligned}\langle g, Rg\rangle - \langle f, Rf\rangle &\overset{\text{(a)}}{\geq} 2\langle Rf, g - f\rangle \overset{\text{(b)}}{=} 2\left\langle B^T p, g - f\right\rangle \\ &= 2\langle p, B(g - f)\rangle = 2\langle p, d - d\rangle = 0.\end{aligned}$$

Here (a) applies convexity of $\varphi(x)$, and (b) substitutes $r = R^{-1}Bp$. ☐

20.4 Effective resistance and the Laplacian

Recall that the Laplacian of a graph G with edge weights $w(e)$ is the symmetric matrix $L : \mathbb{R}^V \to \mathbb{R}^V$ defined by

$$\langle x, Lx \rangle = \sum_{e=\{u,v\} \in E} w(e)(x_u - x_v)^2.$$

Lemma 20.5. *Let $w(e) = 1/r_e$ for all e. Then $L = BR^{-1}B^T$.*

To prove Lemma 20.5, since both matrices are symmetric, it suffices to show that $\langle x, Lx \rangle = \left\langle x, BR^{-1}B^T x \right\rangle$ for all x. We leave this calculation to the reader as exercise 20.3.

Recall that the effective resistance of d is the minimum energy attained by the electrical flow. The effective resistance has the following closed form, drawing a direct connection between the electrical flow and the pseudoinverse of L. Note that $L^{-1}d$ is well-defined because we assume G is connected and d is orthogonal to the kernel of L; i.e., $\mathbb{1}$.

Theorem 20.6. *Given a connected electrical network with resistances r, let L be the Laplacian of the corresponding undirected graph with weights $1/r$. Let d be a fixed set of demands inducing an electrical flow f with electrical potentials p. We have the following.*

1. *$Lp = d$.*
2. *(effective resistance of d) $= \langle d, L^{-1}d \rangle = \langle p, Lp \rangle = \langle p, d \rangle$.*

Proof. Let f be the electrical flow and p the electrical potentials with respect to d. For the first claim, we have

$$Lp = BR^{-1}B^T p = Bf = d.$$

For the second, we have

$$\langle f, Rf \rangle \overset{\text{(a)}}{=} \left\langle R^{-1}B^T p, RR^{-1}B^T p \right\rangle = \langle p, Lp \rangle = \left\langle Lp, L^{-1}Lp \right\rangle \overset{\text{(b)}}{=} \left\langle d, L^{-1}d \right\rangle$$

where (a) is by Ohm's Law. □

20.5 Effective conductance

Consider the following optimization problem.

$$\text{minimize } \langle p, Lp \rangle \text{ over } \langle p, d \rangle = 1. \tag{20.2}$$

In the special case of $d = 1_t - 1_s$, we are seeking the potentials p minimizing $\langle p, Lp \rangle$ subject to s and t being separated by 1 unit. The optimum value to (20.2) is sometimes called the *effective conductance.*

The first order conditions tell us that the optimum solution $p^\star$ satisfies $2Lp^\star = \lambda d$, hence $p^\star = (\lambda/2) L^{-1} d$, for some scalar λ. To identify λ, we plug into $\langle p^\star, d \rangle = 1$: We have

$$1 = \langle p^\star, d \rangle = \frac{\lambda}{2} \left\langle d, L^{-1} d \right\rangle,$$

hence

$$\lambda = \frac{2}{\langle d, L^{-1} d \rangle}.$$

Note that $2/\lambda = \langle d, L^{-1} d \rangle$ is the effective resistance of d. Returning to (20.2), we have

$$\langle p^\star, Lp^\star \rangle = \left\langle Lp^\star, L^{-1} Lp^\star \right\rangle = \frac{\lambda^2}{4} \left\langle d, L^{-1} d \right\rangle = \frac{1}{\langle d, L^{-1} d \rangle}.$$

The following theorem summarizes our developments.

Theorem 20.7. (20.2) *has optimum solution* $p^\star = L^{-1} d / 2 \langle d, L^{-1} d \rangle$ *and optimum value* $1/\langle d, L^{-1} d \rangle$.

In particular we have the following symmetry between the effective resistance and the effective conductance of a demand vector d.

Corollary 20.8. *For any demands d, the effective resistance of d is the reciprocal of the effective conductance of d.*

An alternative interpretation of this symmetry is as follows.

Corollary 20.9. *For any d-flow f, and any potentials p with $\langle p, d \rangle = 1$, we have*

$$\langle f, Rf \rangle \langle p, Lp \rangle \geq 1.$$

The inequality is tight for a unique f and p (modulo $\mathbb{1}$).

20.6 Hitting times and cover times

Let us now return to our original discussion on random walks, where we were particularly interesting in understanding the hitting time and cover times of an undirected graph. We restate their definitions for the reader's convenience.

Definition 20.1. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Consider a random walk starting from a vertex s. The* hitting time *from s to a vertex v, denoted $H(s, v)$ is the expected number of steps until a random walk from s reaches v. The* cover time *from s, denoted $C(s)$, is the expected number of steps until a random walk from s visits every vertex in the graph.*

Fix $t \in V$. We want to analyze the hitting time $H(v, t)$ for all $v \in V$. We have

$$H(t, t) = 0,$$

by definition. For other vertices $v \neq s$, we have

$$H(u, t) = 1 + \frac{1}{\deg(u)} \sum_{(u,v) \in E} H(u, t)$$

by definition of the random walk.

The key idea interpret this system of equations in terms of electrical networks. *Define vertex potentials $p \in \mathbb{R}^V$ by*

$$p_u = H(u, t).$$

The vertex potentials induces a flow $f = Bp$, which more explicitly, carries flow

$$f(u, v) = p_v - p_u.$$

We have $p_t = H(t, t) = 0$. We also have the following equivalent equations for all $u \neq t$.

$$p_u = 1 + \frac{1}{\deg(u)} \sum_{\{(u,v) \in E\}} p_v$$

We can rewrite this as

$$\sum_{\{(u,v) \in E\}} (p_u - p_v) = \deg(u).$$

Recall that $f(u, v) = p_u - p_v$. Thus p *encodes a flow routing* $\deg(u)$ *units of flow out of each $u \neq t$ and* $2m - \deg(t)$ *units of flow into v.* In terms of hitting times, we have shown the following.

Lemma 20.10. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $t \in V$. Let p be the electrical potentials routing the demands*

$$d_u = \begin{cases} -\deg(u) & \text{if } u \neq t \\ 2m - \sum_u \deg(u) & \text{otherwise.} \end{cases}$$

Then for all u,

$$H(u, t) = p_u - p_t.$$

Lemma 20.11. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $s, t \in V$.*

$$H(s, t) + H(t, s) = 2m(\textit{effective resistance from } s \textit{ to } t).$$

Proof. Let $p = H(\cdot, s)$ and $q = H(\cdot, t)$ be the electrical potentials encoding the hitting times to s and to t, respecitviely. Then

$$H(u, v) + H(v, u) = (p_u - p_v) + (q_v - q_u) = r_u - r_v$$

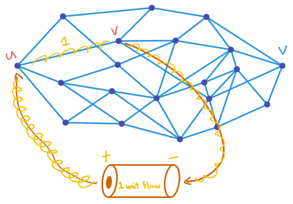
for $r = p - q$. The corresponding flow is the difference between the flows induced by p and q:

$$B^T r = B^T p - B^T q.$$

Then $B^T r$ routes $2m$ units of flow from s to t. This means that $(1/2m)r$ is the electrical potential required to route one unit of flow from s to t. Let d be the demands for routing one unit of flow from s to t. we have

$$\begin{aligned} H(s, t) + H(t, s) = \langle r, d \rangle &= \frac{1}{2m}\left\langle r, BB^T r \right\rangle \\ &= \frac{1}{2m}\left\langle B^T r, B^T r \right\rangle = 2m\langle (1/2m)Br, (1/2m)Br \rangle, \end{aligned}$$

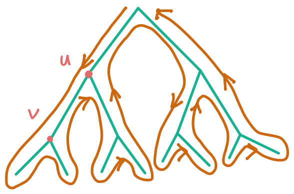
as desired. $\square$



Theorem 20.12. *Let $e = (s,t) \in E$. Then*

$$H(s,t) + H(t,s) \leq 2m.$$

Proof. The flow sending one unit of flow along e is an (s,t)-flow with electric energy 1; the electrical flow is only better. Plugging 1 into Lemma 20.11 as an upper bound on the effective resistance gives the bound we seek. □



Theorem 20.2. *Let $G = (V,E)$ be an undirected graph with m edges and n vertices. Let $s \in V$. Then the cover time $C(s)$ is bounded above by $C(s) \leq m(n-1)$.*

Proof. Fix any spanning tree T, and fix a traversal on T starting and ending at T, which corresponds to a fixed sequence of oriented edges of T, with each edge appearing once in each direction. Imagine trying to simulate this walk randomly: for each edge (u,v) in sequence, starting from u, we do a random walk from u until we hit v. Then we do the same for the next edge in the spanning tree. The expected time to traverse an edge $e = (u,e) \in T$ is $H(u,v)$. The total time over the entire spanning tree is

$$\sum_{e=\{u,v\}\in T} H(u,v) + H(v,u) \leq \sum_{e=\{u,v\}\in T} 2m \leq 2m(n-1).$$

□

20.7 Additional notes and materials

See also [DS84].

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

20.8 Exercises

Exercise 20.1. Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $s, t \in V$ be connected by a path of k edges. Show that

$$H(s,t) \leq km.$$

Exercise 20.2. The goal of this exercise is to understand why we required the graph to be undirected. Design and analyze, for $n \in \mathbb{N}$, an unweighted, strongly connected and directed graph $G = (V, E)$ on n vertices and two vertices $s, t \in V$ where the hitting time from s to t is exponential in n.

Exercise 20.3. Complete the proof of Lemma 20.5.

Exercise 20.4. This model is inspired by the following scenario. Suppose Alice and Bob are separated and lost in a big city. They both decide to start randomly walking around the city. How long does it take, in expectation, for Alice and Bob to meet?

To model this formally, let $G = (V, E)$ be an undirected graph with m edges and n vertices. Alice and Bob start on two vertices s and t, respectively, and they take independent random walk in synchronized fashion. That is, in each step, Alice and Bob independently choose a uniformly random edge incident to their current vertex, and traverse that edge. The random walks end when Alice and Bob meet each other at the same vertex.

Now it might be impossible for Alice and Bob to ever meet. For example, if G consists of a single edge $\{s, t\}$, then Alice and Bob will keep switching vertices and never meet. But suppose we promise that it is possible for Alice and Bob to meet. More specifically, suppose there is a vertex x for which there are ℓ-edge walks from both s to x and from t to x in G. Propose and prove an upper bound, the smaller the better, on the expected number of steps until Alice and Bob meet, as a function of m, n, and possibly ℓ; or alternatively, prove that the expected number of steps is $+\infty$.

Chapter 21

Spectral analysis of undirected random walks

21.1 The Laplacian of a graph

Let $G = (V, E)$ be an undirected graph with positive edge weights $w \in \mathbb{R}^E_{>0}$. In our analysis of electrical networks (Chapter 20), we briefly came across the *Laplacian form* of G. We reintroduce the Laplacian $L : \mathbb{R}^V \to \mathbb{R}^V$ of from a different perspective.

We first show how to model a single edge by a rank-1 matrix; an entire graph is then modeled by the corresponding weighted sum over its edges. We will work in the n-dimensional vector space $\mathbb{R}^V$ – one coordinate per vertex. Edges are modeled as matrices in $\mathbb{R}^{V \times V}$.

The *Laplacian of an (unweighted) edge* $e = \{u, v\}$ is the rank-1 matrix

$$L_e = (1_u - 1_v) \otimes (1_u - 1_v)$$

where $1_u \in \{0,1\}^V$ denotes the indicator vector[1] for u. Here $a \otimes b$ denotes the outer product of two vectors a, b, defined by $\langle x, (a \otimes b) y \rangle = \langle a, x \rangle \langle b, y \rangle$. Note that the expression for L_e is indifferent to whether we wrote $1_u - 1_v$ or $1_v - 1_u$, as long as it is symmetric. For any input vector $x \in \mathbb{R}^V$, we have

$$\langle x, L_e x \rangle = (x_u - x_v)^2.$$

For an undirected graph $G = (V, E)$ with positive edge weights $w : E \to \mathbb{R}_{>0}$, the *Laplacian of the graph* is the weighted sum of Laplacians of its edges,

$$L = \sum_e w(e) L_e.$$

Given an input vector $x \in \mathbb{R}^V$, we have

$$\langle x, Lx \rangle = \sum_{e \in E} w(e) \langle x, L_e x \rangle = \sum_{e=(u,v) \in E} w(e) (x_u - x_v)^2.$$

[1]We are avoiding the conventional notation e_u for the standard basis vectors because e is so frequently used for edges.

That is, the L induces a simple sum of squared differences on x, based on the edges of the graph.

In fact, it induces a very familiar sum of squares. Recall that the the electrical flow problem is to minimize $\langle f, Rf\rangle$ over $f \in \mathbb{R}^E$ s.t. $Bf = d$. Here $R = \operatorname{diag}(r)$ is the diagonal map of resistances $r \in \mathbb{R}^E_{>0}$. $d \in \mathbb{R}^V$ represents the flow demands and $B : \mathbb{R}^E \to \mathbb{R}^V$ maps flows to the net flow at each vertex. We also saw that, by first-order optimality conditions, the electrical flow is always of the form $f = R^{-1}B^{\mathsf{T}}p$ for a set of vertex potentials p. Then we have

$$\langle f, Rf\rangle = \left\langle R^{-1}B^{\mathsf{T}}p, RR^{-1}B^{\mathsf{T}}p\right\rangle = \sum_{e=(u,v)\in E} \frac{1}{r_e}(p_u - p_v)^2.$$

That is, we are choosing p as to minimize the Laplacian of the graph with edge weights $w(e) = 1/r_e$.

The Laplacian L is also closely tied to the cuts of G. Given a set $S \subset V$, if we letting $\mathbb{1}_S$ and $\mathbb{1}_{\bar{S}}$ denote the indicator vectors of S and $\bar{S}$, we have

$$\langle \mathbb{1}_S, L\mathbb{1}_{\bar{S}}\rangle = 4w(\delta(S)).$$

Recall that a linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is *symmetric* if $A = A^{\mathsf{T}}$. It is easy to see that the Laplacian L is symmetric: each L_e is symmetric since in general $(a \otimes b)^{\mathsf{T}} = (b \otimes a)$, and L is a positively weighted combination of L_e's. Another salient property of L is that, as a sum of squares,

$$\langle x, Lx\rangle \geq 0 \text{ for all } x \in \mathbb{R}^n.$$

These two properties make L a member of the following very importance class of linear operators.

Definition 21.1. *A linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is a* positive semi-definite *linear operator if*

(a) A is symmetric.

(b) $\langle x, Ax\rangle \geq 0$ for all $x \in \mathbb{R}^n$.

A is (strictly) positive definite *if in addition to being positive semi-definite,*

(c) A is invertible.

The Laplacian L is *not* invertible: $L\mathbb{1} = \mathbb{0}$. If G is connected, and we restrict to the $n-1$ space $\mathbb{R}^V/\mathbb{1}$, then L is invertible and (strictly) positive definite (see exercise 21.3).

21.2 The Spectral Theorem for Symmetric Maps

Our first discussion on random walks (Chapter 19) showed that eigenvectors give insight into the behavior of a linear map. We will see that the eigenvectors of symmetric linear maps — such as the Laplacian — are particularly well-behaved and useful.

Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. Recall that a vector $x \in \mathbb{R}^n$ is an *eigenvector* of A with *eigenvalue* $\mu_1 \in \mathbb{C}$ of $Ax = \mu_1 x$. Recall the following facts, which apply generally to all linear maps.

Fact 21.2. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. Then A has an eigenvalue $\mu_1 \in \mathbb{C}$ and eigenvector $x \in \mathbb{C}^n$.*

If A is symmetric, then we can strengthen this fact to assert a real-valued eigenvalue and eigenvector with real-valued coordinates, and moreover they can be obtained by optimization.

Lemma 21.3. *Let $L : X \to X$ be a symmetric linear map in a vector space X over $\mathbb{R}$. Let x maximize $\langle x, Lx\rangle$ subject to $\|x\| = 1$. Then $Lx = \mu_1 x$ for $\mu_1 = \langle x, Lx\rangle$.*

Proof. We claim that for any $u \in X$ with $\|u\| = 1$ and $\langle u, x\rangle = 0$, $\langle u, Lx\rangle = 0$. If Lx is orthogonal to u for every u orthogonal to x, then we must have $Lx \in \operatorname{span}(x)$; i.e., $Lx = \mu_1 x$ for some $\mu_1 \in \mathbb{R}$. Upon inspection, $\mu_1 = \mu_1\langle x, x\rangle = \langle x, Lx\rangle$, as claimed.

Let $u \in X$ with $\|u\| = 1$ and $\langle u, x\rangle = 0$. Define

$$f(\epsilon) = \left\langle \frac{x + \epsilon u}{\sqrt{1+\epsilon^2}}, L\left(\frac{x+\epsilon u}{\sqrt{1+\epsilon^2}}\right)\right\rangle = \frac{\langle x + \epsilon u, L(x+\epsilon u)\rangle}{1+\epsilon^2}.$$

$f(\epsilon)$ can be interpreted as perturbing x slightly in the direction of u and renormalizing, and then computing the inner product over L. Note that $\|x + \epsilon u\|^2 = \|x\|^2 + \epsilon^2\|u\|^2 = 1 + \epsilon^2$, so $\frac{x+\epsilon u}{\sqrt{1+\epsilon u}}$ is indeed a normal vector that competes with x in maximizing $\langle x, Lx\rangle$. In particular, by choice of x, $f(\epsilon)$ is maximized at $f(0) = \langle x, Lx\rangle$. Optimality at 0 implies that $f'(0) = 0$. Expanding out $f'(0)$, we find that $\langle u, Lx\rangle = 0$, as desired. (See exercise 21.4). □

Remark 21.4. An alternative proof starts from the fact there exists a complex eigenvalue and eigenvector, and goes on to show that this eigenvalue must be real-valued and that there is a corresponding eigenvector with real-valued coordinates.

The following theorem, called the *spectral theorem* for symmetric operators, strengthens the previous lemma to show that *all* the eigenvalues and eigenvectors are real-valued.

Theorem 21.5. *Let X be an n-dimensional vector space over $\mathbb{R}$. Let $A : X \to X$ be a symmetric linear map. Then there exists an orthonormal basis $v_1, \ldots, v_n$ of X and n scalar values $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ such that*

$$A = \lambda_1(v_1 \otimes v_1) + \lambda_2(v_2 \otimes v_2) + \cdots + \lambda_n(v_n \otimes v_n). \tag{21.1}$$

Proof. If $n = 0$, then the claim is tautological, as X is the trivial vector space $\{\mathbb{0}\}$ and A can be expressed as an empty sum. Suppose $n \geq 1$. By Lemma 21.3, A has a real-valued eigenvalue μ_1 with a corresponding eigenvector $u \in X$. By scaling u, we may assume $\|u\| = 1$. Consider the map $B = A - \mu_1(u \otimes u)$. B is also symmetric, and maps the space $\operatorname{span}(x) = \{\alpha x : \alpha \in \mathbb{R}\}$ to $\mathbb{0}$. Let $Y = \{y \in X : \langle x, y\rangle = 0\}$ be the subspace of X orthogonal to x. We have $\dim(Y) = n - 1$.

We claim that B maps Y into Y. Indeed, for any $y \in Y$, we have

$$\langle x, By\rangle = \langle Bx, y\rangle = \langle \mathbb{0}, y\rangle = 0,$$

so $By \in Y$.

Thus B restricts to a linear and symmetric operator on Y. By induction on n, there is an orthonormal basis $v_1, \ldots, v_{n-1}$ of Y and scalar values $\lambda_1, \ldots, \lambda_{n-1} \in \mathbb{R}$ such that

$$B = \lambda_1(v_1 \otimes v_1) + \cdots + \lambda_{n-1}(v_{n-1} \otimes v_{n-1}).$$

Let $\lambda_n = \mu_1$ and $v_n = u$. Observe that $v_1, \ldots, v_n$ is an orthonormal basis of X. We have

$$\lambda_1(v_1 \otimes v_1) + \cdots + \lambda_n(v_n \otimes v_n) = B + \lambda_n(v_n \otimes v_n) = A,$$

as desired. ☐

Theorem 21.5 makes the structure of any symmetric map $A : \mathbb{R}^n \to \mathbb{R}^n$ extremely transparent. By Theorem 21.5, let $v_1, \ldots, v_n \in \mathbb{R}^n$ and $\mu_1, \ldots, \mu_n \in \mathbb{R}$ be such that

$$A = \mu_1(v_1 \otimes v_1) + \cdots + \mu_n(v_n \otimes v_n).$$

We assume that the μ_i's are in nonincreasing order: $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n$.[2]

For any input vector $x \in \mathbb{R}^n$, we can write x uniquely in the basis $\{v_1, \ldots, v_n\}$ as

$$x = \alpha_1 v_1 + \cdots + \alpha_n v_n,$$

[2]Notationally, we try to use μ_i's when listing the eigenvalues in decreasing order, and λ_i's when listing the eigenvalues in increasing order.

where $\alpha_i = \langle x, v_i \rangle$. Then we have

$$Ax = \mu_1 \alpha v_1 + \cdots + \mu_n \alpha_n v_n.$$

That is, in the basis $\{v_1, \dots, v_n\}$, A simply rescales the ith coordinate by a factor of μ_i. That is to say:

Every symmetric matrix is a diagonal matrix up to rotation (i.e., change in basis).

We can see from the construction in the proof that the μ_i's are the eigenvalues of A and the v_i's are eigenvectors. But this fact is even more obvious in hindsight given the spectral representation (21.1). For each i, we have

$$Av_i = \mu_i (v_i \otimes v_i) v_i = \mu_i v_i,$$

by orthonormality of the v_i's. The following theorem gives a min-max characterization of the eigenvalues and follows immediately from the spectral theorem. It is often called the *Courant-Fischer minimax theorem*.

Theorem 21.6. *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a symmetric linear operator. Let $\mu_1, \dots, \mu_n$ be the n eigenvalues of A (with multiplicity) in decreasing order. Then*

$$\mu_k = \min_{S : \dim(S) = k-1} \max_{x \in X / S} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.$$

21.3 Random walks in undirected graphs

Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let us assume that G is connected. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map of G. Recall that $R = AD^{-1}$, where $A : \mathbb{R}^V \to \mathbb{R}^V$ is the weighted adjacency map and $D = \operatorname{diag}(d)$ is the diagonal map of weighted vertex degrees $d = A\mathbb{1}$. Recall that R can be analyzed by the *Perron-Frobenius theorem*, which for random walks gave us a lot of information about the eigenvalues and eigenvectors of R. In particular, all of the eigenvalues of R lie in the range $[-1, 1]$, and it has eigenvalue 1 with multiplicity 1. There is a strictly positive eigenvector for eigenvalue 1 that defines a unique *stationary distribution*. Before, we proved the existence of a unique stationary distribution for strongly connected *directed* random walk. For undirected random walk, the stationary distribution is very straightforward.

Theorem 21.7. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map of G. Then the stationary distribution of R is proportional to the weighted degrees of its vertices.*

Proof. We already know that the stationary distribution exists and is unique. We have

$$R(A\mathbb{1}) = A(\operatorname{diag}(A\mathbb{1}))^{-1}A\mathbb{1} = A\mathbb{1}.$$

□

While R is not a symmetric map[3], we can extend the spectral theorem for symmetric maps to R by way of *similarity.*

Definition 21.8. *Two linear maps $A, B : X \to X$ are* similar *if $A = C^{-1}BC$ for an invertible map $C : X \to X$.*

Lemma 21.9. *Let A and B be similar. Then their kernels are isomorphic. In particular, if $A = C^{-1}BC$, then C restricts to an isomorphism between* $\ker(A)$ *and* $\ker(B)$.

Lemma 21.10. *Let $A, B : X \to X$ be two linear maps. If A and B are similar, then A and B have the same eigenvalues with the same multiplicities. If $A = C^{-1}BC$, then C maps eigenvectors of A to eigenvectors of B with the same eigenvalues.*

Proof. For all λ, $A - \lambda I$ and $B - \lambda I$ are also similar. □

We introduce the *normalized walk matrix* as the map $Q : \mathbb{R}^V \to \mathbb{R}^V$ defined by

$$Q = D^{-1/2}RD^{1/2} = D^{-1/2}AD^{-1/2}.$$

By the first equality above, Q is similar to R, and thus has all its eigenvalues in the range $[-1, 1]$ and eigenvalue 1 with multiplicity 1. On the other hand, by the second equality, Q *is symmetric.* As such, it has an orthonormal basis of eigenvectors. Let $1 = \mu_1, \ldots, \mu_n \geq -1$ list the eigenvalues of Q in decreasing order. Let $u_1, \ldots, u_n$ be an orthonormal basis of $\mathbb{R}^V$ such that

$$Q = u_1 \otimes u_1 + \mu_2(u_2 \otimes u_2) + \cdots + \mu_n(u_n \otimes u_n).$$

(Here we substituted $\mu_1 = 1$). We also know, from Lemma 21.10, that $D^{1/2}u_1$ must correspond to the uniform distribution, $d/\langle \mathbb{1}, d\rangle$. Since u_1 has unit length, we have

$$u_1 = \frac{D^{-1/2}(d)}{\|D^{-1/2}d\|} = \frac{1}{\sqrt{2W}}\sqrt{d},$$

[3]unless G is regular; see exercise 21.5

where $\sqrt{d}$ represents the entrywise square root of d, and $W = \sum_{e \in E} w(e) = \frac{1}{2} \sum_{v \in V} d(v)$ is the sum of all edge weights. Substituting back in, we have

$$Q = \frac{1}{2W}\left(\sqrt{d} \otimes \sqrt{d}\right) + \mu_2(u_2 \otimes u_2) + \cdots + \mu_n(u_n \otimes u_n).$$

Let us now consider the *convergence rate* of a random walk. Let $x \in \Delta^V$ be any initial probability distribution over Q. We want to understand the distribution $R^k x$ obtained after k steps of the random walk. Observe first that

$$\begin{aligned} R^k x &\overset{\text{(a)}}{=} D^{1/2} Q^k D^{-1/2} x \\ &= D^{1/2}\left(\frac{1}{2W}\left(\sqrt{d} \otimes \sqrt{d}\right) + \mu_2(u_2 \otimes u_2) + \cdots + \mu_n(u_n \otimes u_n)\right)^k D^{-1/2} x \\ &\overset{\text{(b)}}{=} D^{1/2}\left(\frac{1}{2W}\left(\sqrt{d} \otimes \sqrt{d}\right) + \mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n)\right) D^{-1/2} x \\ &\overset{\text{(c)}}{=} \frac{d}{2W} + \left(\mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n)\right) D^{-1/2} x \end{aligned}$$

(a) substitutes in $R = D^{1/2} Q D^{-1/2}$, where the $D^{-1/2}$ and $D^{1/2}$ terms between Q's cancel out. (b) is because the u_i's are othonormal[4] – here we see some of the power of the spectral theorem! (c) is because

$$D^{1/2}\left(\sqrt{d} \otimes \sqrt{d}\right) D^{-1/2} x = (d \otimes \mathbb{1}) x = \langle \mathbb{1}, x \rangle d = d.$$

Consider the RHS of the last equation above. Remarkably, the stationary distribution, $d/2W$, has emerged, followed by a messy term involving the non-dominant eigenvalues and eigenvectors. Thus the difference between $R^k x$ and the stationary distribution is precisely

$$D^{1/2}\left(\mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n)\right) D^{-1/2} x.$$

Let $S = \mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n)$; S is symmetric, with eigenvalues $0, \mu_2^k, \ldots, \mu_n^k$.

[4]We should point out that $(a \otimes b)(c \otimes d) = \langle b, c \rangle (a \otimes d)$

Let $\Delta_{\max}$ be the maximum degree in G and let $\Delta_{\min}$ be the minimum degree.

$$\begin{aligned}
\left\| R^k x - \frac{1}{2W} d \right\|^2 &= \left\| D^{1/2} S D^{-1/2} x \right\|^2 \\
&= \left\langle S D^{-1/2} x, D\left(S D^{-1/2} x\right) \right\rangle \\
&\overset{\text{(d)}}{\leq} \Delta_{\max} \left\| S D^{-1/2} x \right\|^2 \\
&= \Delta_{\max} \left\langle D^{-1/2} x, S^2 \left(D^{-1/2} x\right) \right\rangle \\
&\overset{\text{(e)}}{\leq} \max\left\{ \mu_2^{2k}, \mu_n^{2k} \right\} \Delta \left\langle x, D^{-1} x \right\rangle \\
&\leq \max\left\{ \mu_2^{2k}, \mu_n^{2k} \right\} \frac{\Delta_{\max}}{\Delta_{\min}}.
\end{aligned}$$

(d) and (e) both follow from the fact that for a symmetric map A with maximum eigenvalue μ_1, we have $\langle x, Ax \rangle \leq \mu_1 \|x\|^2$ for all x (Lemma 21.3). D has maximum eigenvalue $\Delta_{\max}$, S^2 has maximimum eigenvalue $\max\left\{ \mu_2^{2k}, \mu_n^{2k} \right\}$, and D^{-1} has maximum eigenvalue $1/\Delta_{\min}$. Recall that $\mu_2, \mu_n \in [-1, 1)$. If μ_2 and μ_n are both bounded away from both 1 and -1, then $\max\left\{ \mu_2^{2k}, \mu_n^{2k} \right\} = \max\{\mu_2, |\mu_n|\}^{2k} \to 0$ as $k \to \infty$. To this end, the *spectral gap* of a random walk R is defined as the difference

$$\gamma = 1 - \max\{\mu_2, |\mu_n|\},$$

where μ_2 is the second largest eigenvalue and μ_n is the smallest eigenvalue. We have given the following bound on the convergence rate as a function of the spectral gaph.

Theorem 21.11. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let G be connected. Let $d \in \mathbb{R}^V_{>0}$ be the weighted degrees of the vertices. Let $\Delta_{\max} = \max_v d(v)$ be the maximum weighted degree and let $\Delta_{\min} = \min_v d(v)$ be the minimum weighted degree. Let $W = \sum_{e \in E} w(e)$ be the sum of edge weights. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map of G and let γ be the spectral gap of R.*

For any initial distribution $x \in \Delta^V$, x converges to the stationary distribution, $s = d/2W$, at a rate of

$$\left\| R^k x - s \right\| \leq (1 - \gamma)^k \sqrt{\Delta_{\max}/\Delta_{\min}}.$$

21.4 Additional notes and materials

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

21.5 Exercises

Exercise 21.1. Let $A \in \mathbb{R}^{n \times n}$. Prove or disprove: A is symmetric iff $\langle x, Ax \rangle = \left\langle x, A^{\mathsf{T}} x \right\rangle$ for all x.

Exercise 21.2. Let $A \in \mathbb{R}^{n \times n}$ be positive semi-definite. Prove or disprove: A is positive definite iff $\langle x, Ax \rangle > 0$ for all $x \neq \mathbb{0}$.

Exercise 21.3. Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of G. Prove that G is connected iff for any $x \notin \text{span}(\mathbb{1})$, we have $\langle x, Lx \rangle > 0$.

Exercise 21.4. Finish the proof of Lemma 21.3, by deriving the derivative $f'(\epsilon)$ and showing that $f'(0) = 0$ implies that $\langle x, Lu \rangle = 0$. Where do we use the assumption that L is symmetric?

Exercise 21.5. Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map. Prove that R is symmetric iff G is regular[5].

Exercise 21.6. Let $K_n = (V, E)$ denote the unweighted complete graph over n vertices, and let $R \in \mathbb{R}^{V \times V}$ denote the random walk matrix for K_n. Calculate all the eigenvalues (along with their multiplicities) of R and calculate the spectral gap. Show your work and explain your reasoning.

Exercise 21.7. Suppose your goal is to start at a random walk at a single vertex and converge to the stationary distribution as fast as possible. Show that one can choose a vertex v such that, starting from an initial distribution of $x = 1_v$, the ℓ_2-distance from the stationary distribution after k steps is at most $(1 - \gamma)^k$.

[5]A graph is regular if every vertex has the same weighted degree

Exercise 21.8. Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of G. Suppose the spectral gap γ is at least some constant, say $\gamma = 1/2$. (Such a graph is called an *expander*).

1. Show that the diameter of G is at most $O(\log n)$.

2. Recall the (s, t)-connectivity problem for which we showed that a random walk gives a $O(\log n)$-space algorithm. Suppose also that G is has constant maximum degree (say, maximum degree 42). Give a *deterministic*, polynomial time, $O(\log n)$-space algorithm for (s, t)-connectivity on G.

Chapter 22

Conductance

22.1 Introduction

Recall our randomized construction of a constant degree expander, as the union of a constant number of uniformly random matchings. With high probability this produces an expander. But given such a randomized graph, how can we verify and know for certain that it has constant expansion? We can obtain a $O(\log n)$-approximation by sparsest cut, but this approximation bound is too rough to decide if the expansion is closer to 1 or $1/\log(n)$.

We will present an algorithm that can certify a constant degree expander. The algorithm is really an approximation algorithm for the *conductance* of a graph, a different but related notion to sparsity that coincides with sparsity for constant degree graphs. The approximation bound we get will be somewhat unusual but it will suffice for constant expansion.

Just as important is *how* we approximate the conductance. We model the input graph as a symmetric matrix called the "Laplacian", and study its eigenvalues. There turns out to be a strong relation between the second smallest eigenvalue and the conductance via what is called "Cheeger's inequality". The Laplacian has many other applications and we will discuss more in later chapters. The general study of graphs via their Laplacian's is called *spectral graph theory* and this approach has yielded many exciting algorithms.

22.2 Sparse cuts

Recall that the *sparsity* of a cut $\delta(S)$ (where $S \subset V$), denoted $\Phi(S)$, is the ratio

$$\Phi(S) \overset{\text{def}}{=} \frac{\bar{w}(\delta(S))}{\min\{|S|, |\bar{S}|\}},$$

where $\bar{S} = V \setminus S$ is the complement and $\bar{w}(\delta(S)) = \sum_{e \in \delta(S)} w(e)$ is the weight of the cut. The sparsity of the graph G is define as the sparsity of its sparsest cut,

$$\Phi(G) \overset{\text{def}}{=} \min_{S \subset V} \Phi(S).$$

To relate sparsity to the Laplacian, note that for any nonempty set $S \subset V$ with at most $n/2$ vertices, we have

$$\Phi(S) = \frac{\bar{w}(\delta(S))}{|S|} = \frac{\langle \mathbb{1}_S, L \mathbb{1}_S \rangle}{\langle \mathbb{1}_S, \mathbb{1}_S \rangle}, \tag{22.1}$$

where $\mathbb{1}_S$ is the $\{0,1\}$-indicator vector for S.

Now we make a deeper connection to the eigenvectors of L. As a positive semidefinite matrix L, L has nonnegative eigenvalues. Moreover, we know that $\mathbb{1}$ is an eigenvector with eigenvalue of 0 – this gives us our smallest eigenvalue. The eigenvector corresponding to the second smallest eigenvalue, denoted λ_2, is given by

$$\lambda_2 = \min_{x : \langle \mathbb{1}, x \rangle = 0} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.$$

Now, consider any cut $\mathbb{1}_S$, and let x be the orthogonal projection from $\mathbb{1}$; namely,

$$x = \mathbb{1}_S - \alpha \mathbb{1} \text{ where } \alpha = \langle \mathbb{1}_S, \mathbb{1} \rangle / \langle \mathbb{1}, \mathbb{1} \rangle = |S|/n$$

Observe that

$$\langle x, x \rangle = \langle x, \mathbb{1}_S \rangle = (1 - \alpha)|S| = (n - |S|)|S|/n.$$

Thus

$$\lambda_2 \leq \frac{\langle x, Lx \rangle}{\langle x, x \rangle} = n \frac{\sum_{e \in \delta(S)} w(e)}{|S|(n - |S|)} \leq 2\Phi(S).$$

Taking the minimum over all sets S, we obtain the following.

Theorem 22.1. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of G. Let λ_2 be the second smallest eigenvalue of L and let $\Phi(G)$ be the sparsity of G. Then*

$$\lambda_2 \leq 2\Phi(G).$$

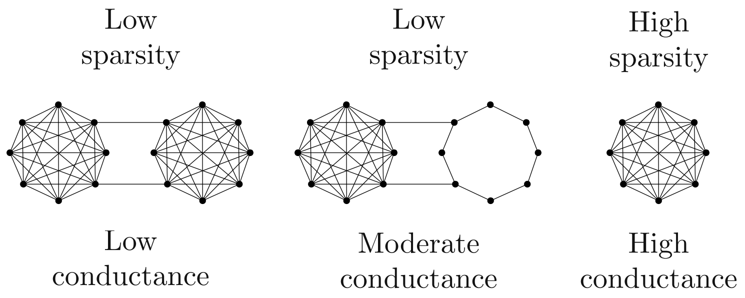


Figure 22.1: Examples of graphs with varying levels of sparsity and conductance.

22.3 Conductance

We now turn to an alternative to sparsity called the *conductance*. For a set of vertices S, the *volume* of S, denoted $\text{vol}(S)$, is the sum of degrees of vertices in S:

$$\text{vol}(S) = \sum_{v \in S} \deg(v).$$

The *conductance* of a set S, denoted $\Psi(S)$, is defined as

$$\Psi(S) = \frac{\bar{w}(\delta(S))}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}$$

Note that $\Psi(S)$ is always positive (for a connected graph) and at most 1. There is a clear resemblance between conductance and sparsity except here the vertices in the denominator are weighted by their degree. Similary to sparsity, we define the conductance of a graph as the minimum conductance of any cut:

$$\Psi(G) = \min_{\emptyset \subsetneq S \subsetneq V} \Psi(S).$$

Like sparsity, conductance is also useful for divide and conquer. The sparsest cut is more suited for divide and conquer on vertices, while conductance, where vertices are weighted by their degree, is more conducive to divide and conquer on edges. Recall that sparsity was naturally motivated by its connection to multicommodity flow. On the other hand, conductance is strongly connected to random walks. Indeed, for any set S, the stationary distribution is in S with probability proportional to $\text{vol}(S)$. Moreover, the conductance of a (small) set S is the amount of probability mass that enters and leaves S in each step at the stationary distribution. Figure 22.1 gives some examples of graphs with different levels of sparsity and conductance.

We would like to express conductance in algebraic terms, similar to sparsity in (22.1). While the numerator in (22.1) seems appropriate, the denominator does not capture the volume. Instead, consider the following quotient:

$$\frac{\langle x, Lx\rangle}{\langle x, Dx\rangle} \text{ where } x \in \mathbb{R}^V. \tag{22.2}$$

For any set S with at most half the total volume, we have

$$\Psi(S) = \frac{\bar{w}(\delta(S))}{\operatorname{vol}(S)} = \frac{\langle \mathbb{1}_S, L\mathbb{1}_S\rangle}{\langle \mathbb{1}_S, D\mathbb{1}_S\rangle}.$$

That said, the quotient (22.2) does not have a direct connection to the Laplacian L in the same way as sparsity did. However, it is connected to the *normalized Laplacian*, which is the map $M : \mathbb{R}^V \to \mathbb{R}^V$ defined by

$$M \stackrel{\text{def}}{=} D^{-1/2} L D^{-1/2}.$$

For any vector x, letting $y = D^{1/2}x$, we have

$$\frac{\langle x, Lx\rangle}{\langle x, Dx\rangle} = \frac{\langle y, My\rangle}{\langle y, y\rangle}.$$

Since the normalized Laplacian M is also symmetric, the RHS models the eigenvalues of M. In today's discussion, we will study the eigenvalues of M and relate it to the condutance of the graph.

We first point out that there are some similarities (in the linear-algebraic sense) to other matrices that we have studied. Let $R = AD^{-1} : \mathbb{R}^V \to \mathbb{R}^V$ denote the random walk map. We define the *normalized random walk matrix* Q as

$$Q \stackrel{\text{def}}{=} D^{-1/2} R D^{1/2} = D^{-1/2} A D^{-1/2}.$$

To draw the connection to M, if we expand $L = D - A$, then we have

$$M = D^{-1/2}(D - A)D^{-1/2} = I - Q = D^{-1/2}(I - R)D^{1/2}.$$

Theorem 22.2. *Let $G = (V, E)$ be an undirected graph with m edges and n vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $M : \mathbb{R}^V \to \mathbb{R}^V$ be the normalized Laplacian and $R : \mathbb{R}^V \to \mathbb{R}^V$ the random walk matrix. Then M is similar to $I - R$, and (equivalently) $I - M$ is similar to R.*

Recall that similarity preserves eigenvalues. Since R has its eigenvalues in $[-1, 1]$ and 1 with multiplicity 1, M has its eigenvalues in $[0, 2]$ and eigenvalue 0 with multiplicity 1. M has eigenvalue 2 iff R has eigenvalue -1.

Cheeger's inequality. We now relate the eigenvalues of M to the conductance of G. The following inequality is called Cheeger's inequality due to an analogous bound by Jeff Cheeger for continuous manifolds.

Theorem 22.3. *Let M be the normalized Laplacian of an undirected graph G, and let λ_2 be the second smallest eigenvalue of M. Then*

$$\frac{\lambda_2}{2} \leq \Psi(G) \leq \sqrt{2\lambda_2}.$$

The presence of the $\sqrt{\cdots}$ on the RHS is unusual for us, who are more used to multiplicative approximations. (Indeed, the square-root leads to a lot of tricky situations, such as the small set expansion hypothesis.)

That said, consider constant degree expanders. In constant degree expanders, the conductance equals the sparsity up to a constant, and where we are interested in sparsity/conductance greater than some constant. Up to now, we did not have an algorithm to verify if a constant degree graph is expander. In this regime, Cheeger's inequality implies that the expansion and λ_2 are within a constant. An algorithmic proof of Theorem 22.3 will give the verification algorithm we seek.

We note that the LHS, $\lambda_2 \leq 2\Psi(G)$, is more straightforward than the RHS, and left to the reader in exercise 22.1. The harder inequality, $\Psi(G) \leq \sqrt{2\lambda_2}$, is proven momentarily in Section 22.4.

Implications for mixing time. Cheeger's inequality allows us to connect the mixing time of a random walk on G to the conductance of G. At a high level, we have established connections between:

- The conductance of G and the second smallest eigenvalue of M.
- The second smallest eigenvalue of M and the second largest eigenvalue of the random walk.
- The second largest eigenvalue of the random walk and the convergence rate to the stationary distribution.

The third connection is a little flimsy, however, because the convergence rate of a random walk on G is determined by the second largest *absolute* value of the eigenvalues of the random work. This can be the smallest (most negative) eigenvalue of the random walk matrix when if it is very close to -1. So instead we analyze the closely related *lazy* random walk where this exception does not occur.

Theorem 22.4. *Let G be an undirected graph with conductance Ψ and lazy random walk matrix S. Then the lazy random walk has spectral gap $\gamma_S \geq \Psi^2/4$, and therefore converges to the stationary distribution at a rate of*

$$\left\| S^t x - s \right\| \leq \exp\left(-t\Psi^2/4\right)\sqrt{\frac{\Delta_{\max}}{\Delta_{\min}}}$$

for any initial distribution x.

Proof. Let S denote the lazy random walk matrix. We have $S = I/2 + R/2$, hence $\mu_k(S) = 1/2 + \mu_k(R)/2$ for all k.[1] In particular, since $\mu_n(S) \geq -1/2$, S has spectral gap

$$\gamma_S = \min\{1 - \mu_2(S), 1 + \mu_n(S)\} \geq \frac{1}{2}\min\{1 - \mu_2(R), 1\}.$$

Furthermore, by similarity of $I - R$ and M, followed by Cheeger's inequality, we have

$$1 - \mu_2(R) = \lambda_2(I - R) = \lambda_2(M) \geq \Psi^2/2,$$

hence $\gamma_S \geq \min\{\Psi^2/2, 1\}/2 = \Psi^2/4$, as desired. □

22.4 Proving $\Psi(G) \leq \sqrt{2\lambda_2}$

In this section we present a proof of the upper bound, $\Psi(G) \leq \sqrt{2\lambda_2}$, based on the proof in [Chu97]. The proof also entails an algorithm due to [Fie73] producing a cut with conductance at most $\sqrt{2\lambda_2}$. Besides the surprising connection to the eigenvalues of M, the algorithm is simple and practical. Based on previous discussions, the reader might be able to guess it.

Recall that

$$\lambda_2 = \min_{y:\left\langle\sqrt{d},y\right\rangle=0} \frac{\langle y, My\rangle}{\langle y, y\rangle} = \min_{x:\langle d,x\rangle=0} \frac{\langle x, Lx\rangle}{\langle x, Dx\rangle}.$$

Let $x \in \mathbb{R}^V$ with $\langle d, x\rangle = 0$ attain λ_2 on the RHS. (We note that eigenvectors, hence x, can be computed.) x is orthogonal to d and, assuming that we have normalized x such that $\langle x, Dx\rangle = 1$, x has a "fractional cut value" of $\langle x, Lx\rangle = \lambda_2$. Our goal is to "round" the "fractional cut" $x \in \mathbb{R}^V$ to a set S without loosing too much on the conductance. How?

[1] Here $\mu_k(A)$ denotes the kth largest eigenvalue of A, and $\lambda_k(A)$ denotes the kth smallest eigenvalue of A.

As an additional hint, we point out that a similar setup arose before for minimum (s,t)-cut and sparsest cut. In each case we had a "fractional cut" from the LP and wanted to produce a discrete one.

Déjà vu! We round x to a cut by outputting the *best cut along the line embedding* x! This is called *Fiedler's algorithm* and pseudocode is given in Fig. 22.2.

Analysis. We first prove Cheeger's inequality, and extract the algorithm from the proof in hindsight. Call a set $S \subseteq V$ *small* if $\operatorname{vol}(S) \leq \operatorname{vol}(\bar{S})$. That is, $\Psi(S) = w(\delta(S))/\operatorname{vol}(S)$ for small S. The following lemma proves Cheeger's inequality for the special case where x is nonnegative with small support.

Lemma 22.5. *Let $x \in \mathbb{R}^V_{\geq 0}$ be nonnegative and $S = \operatorname{support}(x)$. If S is small, then*

$$\langle x, Lx\rangle \geq \frac{\Psi^2}{2}\langle x, Dx\rangle.$$

Proof. Let $k = |S|$, and enumerate $S = \{v_1, \dots, v_k\}$ in decreasing order of $x(v_i)$. For each i, let $S_i = \{v_1, \dots, v_i\}$. All S_i are small. We have

$$\begin{aligned}
\Psi\langle x, Dx\rangle &= \Psi \sum_{i=1}^{k} x^2(v_i)\deg(v_i) = \Psi \sum_{i=1}^{k} x^2(v_i)(\operatorname{vol}(S_i) - \operatorname{vol}(S_{i-1})) \\
&\overset{\text{(a)}}{=} \Psi \sum_{i=1}^{k-1} \operatorname{vol}(S_i)\Big(x^2(v_i) - x^2(v_{i+1})\Big) \\
&\overset{\text{(b)}}{\leq} \sum_{i=1}^{k-1} \Big(x^2(v_i) - x^2(v_{i+1})\Big) w(\delta(S_i)) \\
&= \sum_{e=uv} w(e)\Big|x^2(u) - x^2(v)\Big| \\
&\overset{\text{(c)}}{\leq} \sqrt{\langle x, Lx\rangle \sum_{e=uv} w(e)(x(u)+x(v))^2} \\
&\overset{\text{(d)}}{\leq} \sqrt{2\langle x, Lx\rangle\langle x, Dx\rangle}.
\end{aligned}$$

(a) interchanges sums (where we dropped the term for $\operatorname{vol}(S_0) = 0$). (b) is by definition of Ψ. (c) is by Cauchy-Schwartz. (d) is by the inequality $(a+b)^2 \leq 2(a^2+b^2)$. ☐

Recall that the algorithm first computes x such that $\langle x, Lx\rangle = \lambda_2\langle x, Dx\rangle$. If x was nonnegative with small support, then applying Lemma 22.5 to x gives $\lambda_2 \geq \Psi^2/2$, as desired. Of course x is neither nonnegative nor supported by a small set of vertices. We first address the issue of nonnegativity.

```
Fiedler($G = (V,E)$, $w \in \mathbb{R}^E_{\geq 0}$)
1. Let $x$ minimize $\frac{\langle x, Lx\rangle}{\langle x, Dx\rangle}$ s.t. $\langle x, d\rangle = 0$ and $x \neq \mathbb{0}$.
2. Number the vertices $v_1, \dots, v_n$ in order of $x(v_i)$. $S_i = \{v_1, \dots, v_i\}$ for all $i$. Return the set
   $S_i$ of minimum conductance.
```

Figure 22.2: Fiedler's algorithm for low-conductance cuts.

Lemma 22.6. *Let $x \in \mathbb{R}^V$, and split $x = x_+ - x_-$ where $x_+, x_- \in \mathbb{R}^V_{\geq 0}$ are nonnegative vectors whose nonzeroes are the positive and the (absolute values of the) negative coordinates of x, respectively. Then $\langle x, Lx\rangle \geq \langle x_+, Lx_+\rangle + \langle x_-, Lx_-\rangle$ and $\langle x, Dx\rangle \leq \langle x_+, Dx_+\rangle + \langle x_-, Dx_-\rangle$.*

Proof. For the first inequality, we have

$$\begin{aligned}\langle x, Lx\rangle &= \sum_{e=\{u,v\}} w(e)(x(u) - x(v))^2 \\ &= \sum_{e=\{u,v\}} w(e)(x_+(u) - x_+(v) - (x_-(u) - x_-(v)))^2 \\ &= \langle x_+, Lx_+\rangle + \langle x_-, Lx_-\rangle - 2\sum_{e=\{u,v\}} w(e)(x_+(u) - x_+(v))(x_-(u) - x_-(v)) \\ &\geq \langle x_+, Lx_+\rangle + \langle x_-, Lx_-\rangle.\end{aligned}$$

For the last inequality, observe that $(a_+ - b_+)(a_- - b_-) \leq 0$ for any real numbers a and b.[2] The second inequality is simpler and the proof is left to the reader. □

Now we can split x into two nonnegative vectors x_+ and x_-. However x_+ and x_- may not have small support. This is addressed by the following lemma that allows us to translate x before splitting into the positive and negative parts.

Lemma 22.7. *Let $\langle x, d\rangle = 0$. For all $\alpha \in \mathbb{R}$,*

$$\frac{\langle x + \alpha\mathbb{1}, Lx + \alpha\mathbb{1}\rangle}{\langle x + \alpha\mathbb{1}, Dx + \alpha\mathbb{1}\rangle} \leq \frac{\langle x, Lx\rangle}{\langle x, Dx\rangle}.$$

Proof. We have $\langle x, Lx\rangle = \langle x + \alpha\mathbb{1}, L(x + \alpha\mathbb{1})\rangle$ because $\mathbb{1} \in \ker(L)$. Meanwhile, consider the function

$$f(\alpha) = \langle x + \alpha\mathbb{1}, D(x + \alpha\mathbb{1})\rangle \overset{\text{(a)}}{=} \langle x, Dx\rangle + \alpha^2\langle d, \mathbb{1}\rangle,$$

where for (a) we recall that $\langle d, x\rangle = 0$. Of course the RHS is minimized at $\alpha = 0$. □

[2] WLOG $a \geq b$. Then $a_+ - b_+ \geq 0$ and $a_- - b_- \leq 0$.

To complete the proof, let $x \in \mathbb{R}^V$ with $\langle x, d \rangle = 0$ and $\langle x, Lx \rangle = \lambda_2 \langle x, Dx \rangle$. Then

$$\langle x, Lx \rangle \geq \langle x_+, Lx_+ \rangle + \langle x_-, Lx_- \rangle \geq \frac{\Psi^2}{2}(\langle x_+, Dx_+ \rangle + \langle x_-, Dx_- \rangle) \geq \frac{\Psi^2}{2}\langle x, Dx \rangle.$$

Rearranging gives

$$\lambda_2 = \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} \geq \frac{\Psi^2}{2},$$

as desired.

Making the proof algorithmic. Let α denote the minimum conductance of all the cuts considered by the algorithm. We can easily adjust the key lemma, Lemma 22.5, to incorporate α, as follows.

Lemma 22.8. *Let $x \in \mathbb{R}^V_{\geq 0}$ be nonnegative with small support $S = \text{support}(x)$. Let $v_1, \dots, v_k$ number S in nonincreasing order of $x(v_i)$, and let $S_i = \{v_1, \dots, v_i\}$ for all i. If all S_i have conductance at least α, then $\langle x, Lx \rangle \geq \alpha^2 \langle x, Dx \rangle / 2$.*

Repeating the same proof as before, except with Lemma 22.8 in place of Lemma 22.5, gives $\lambda_2 \geq \alpha^2/2$, hence $\alpha \leq \sqrt{2\lambda_2}$.

22.5 Additional notes and materials

We refer the reader to [Tre16; Spi19] for more on spectral graph theory.

Spring 2024 lecture notes. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

22.6 Exercises

Exercise 22.1. Here we prove the lower bound in Cheeger's inequality (Theorem 22.3), $\Psi(G) \geq \lambda_2/2$. Let $S \subseteq V$ induced the minimum conductance cut; i.e., $\text{vol}(S) \leq \text{vol}(V)/2$ and $\Psi(G) = \Psi(S)$. Consider the vector $x = D^{1/2}\mathbb{1}_S$ and let $y = D^{1/2}\mathbb{1}_{\bar{S}}$.

1. Show that
$$\frac{\langle x, Mx\rangle}{\langle x, x\rangle} = \frac{\langle \mathbb{1}_S, L\mathbb{1}_S\rangle}{\langle \mathbb{1}_S, D\mathbb{1}_S\rangle}, \; \frac{\langle y, My\rangle}{\langle y, y\rangle} = \frac{\langle \mathbb{1}_{\bar{S}}, L\mathbb{1}_{\bar{S}}\rangle}{\langle \mathbb{1}_{\bar{S}}, D\mathbb{1}_{\bar{S}}\rangle}, \text{ and } \langle x, y\rangle = 0.$$

2. Show that for any $\alpha, \beta \neq 0$, we have
$$\frac{\langle \alpha x + \beta y, M(\alpha x + \beta y)\rangle}{\langle \alpha x + \beta y, \alpha x + \beta y\rangle} \leq 2\Psi(G).$$

3. Argue that one can choose $\alpha, \beta \neq 0$ such that $\left\langle \sqrt{d}, \alpha x + \beta y\right\rangle = 0$.

4. Finally, prove that the second smallest eigenvector of M is at most $2\Psi(G)$.

Chapter 23

Deterministic log-space connectivity

23.1 Introduction

Consider the (s,t)-connectivity problems in undirected graphs. Let $G = (V,E)$ be an undirected graph with m edges and n vertices. Let $s,t \in V$. We want to know if s and t are connected in G. This is very easy in normal settings, but in *logarithmic space*, we cannot mark the vertices as we visit them. Previously we solved this with random walks. The algorithm takes a random walk from s and answers yes if we reach t within the first $O(mn)$ steps. The algorithm was justified by analyzing the *cover time*: we showed that a random walk from s visits *every* vertex (connected to s) in $\leq 2mn$ steps in expectation. A follow up exercise showed that if s and t are connected by a path of k edges, then the expected number of steps is $O(mk)$.

We also know that random walks have stationary distributions, which arise as the unique eigenvector of eigenvalue 1 of the random walk map $R : \mathbb{R}^V \to \mathbb{R}^V$. For undirected graphs, the convergence rate is connected to the *spectral gap* γ of the random walk matrix. The spectral gap γ is the difference between the maximum eigenvalue 1 and the absolute value of any other eigenvalue of the random walk matrix.

Let $d \in \mathbb{N}^V$ be the degrees of G, and recall that the stationary distribution (for undirected graphs) is proportional to $d/2m$. Let $x_k \in \Delta^V$ denote the distribution after k random steps (from some arbitrary initial distribution $x_0 \in \Delta^V$). Then x_k converged to the stationary distribution s at a rate of

$$\|x_k - d/2m\|_2 \leq (1-\gamma)^k \sqrt{n}.$$

An undirected graph is called an *expander graph* if the spectral gap γ is at least a constant. Suppose G is an expander with spectral gap (say) 1/2. Then a random walk converges exponentially fast at a rate of

$$\|x_k - d/2m\|_2 \leq 2^{-k} \sqrt{n}.$$

For $k = O(\log n)$ steps, we have

$$\|x_k - d/2m\|_\infty \leq \|x_k - d/2m\|_2 \leq \frac{n}{2^k}.$$

Since $d(v)/2m \geq 1/2m$ for all v, we have $x_k(v) > 0$ for $k \geq O(\log n)$.

This holds for any initial distribution, so suppose we started a single vertex s. Then $x_k(v) > 0$ implies that *there exists a path form s to v of length k.* That is, any graph with constant spectral gap has $O(\log n)$ diameter.

Now suppose that G also had maximum degree at most (say) 8. A random walk on G only needs to sample $O(1)$ random bits to sample each random step. If there is a path from s to t of length $O(\log n)$, then there is some sample of $O(\log n)$ random bits that generates a random walk from s to t. The number of random bits is so little that we can deterministically enumerate and try all $2^{O(\log n)} = \text{poly}(n)$ possible bit strings!

Observation 23.1. *(s,t)-connectivity in a constant degree expander can be decided* deterministically *in* poly(n) *time and* $O(\log n)$ *space.*

Our goal is to generalize this argument from expanders to general graphs. We will prove the following theorem due to Reingold [Rei08].

Theorem 23.2 ([Rei08])**.** *There is a $O(\log n)$ space, polynomial time deterministic algorithm for (s,t)-connectivity in undirected graphs with n vertices.*

23.2 An overview of the deterministic connectivity algorithm

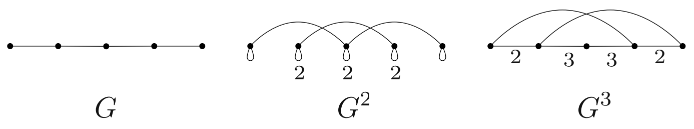
We first give a high level of the algorithm and analysis of Theorem 23.2. We will introduce some important technical lemmas, and show how to complete the proof of Theorem 23.2 assuming they hold true. We will prove the lemmas in subsequent sections.

The approach taken by [Rei08] starts from Observation 23.1, that deterministic log-space connectivity is easy on expander graphs. The high-level idea is to implicitly convert the input graph G into a constant degree expander. [Rei08], building on previous work [RVW00], applies a sequence of graph transformations to G that iteratively improves the spectral gap. These transformations are done implicitly because in $O(\log n)$ space we cannot explicitly rebuild the graph. One can then derandomize the random walk algorithm by enumeration on this artificial expander graph.

We assume the graph G is a regular graph where all vertices have degree d. It is not difficult to take the input graph and transform it into a regular graph; we leave this as an exercise to the reader. The graph transformations introduced below preserve the regularity of the graph (though d changes). For the rest of this section, we always let G denote an a regular undirected graph with n vertices and degree d.

We now introduce the two graph transformations that we will use.

Graph transformation 1: Powering.



For $k \in \mathbb{N}$, the *kth power of G*, denoted G^k, is the multi-graph on V with edges corresponding to all the k-step walks for G. That is, for $u, v \in V$, the edge (u, v) has multiplicity equal to the number of walks from u to v.

If G has random walk matrix R, then G^k has random walk matrix R^k, and the eigenvalues of R^k raise the eigenvalues of R to the kth power. This improves the spectral gap which is good. Unfortunately powering also increases the maximum degree.

Lemma 23.3. *G^k is a regular undirected graph on V with degree d^k and random walk map R^k. If R has spectral gap γ, then R^k has spectral gap $1 - (1 - \gamma)^k$.*

The proof is left to the reader.

Graph transformation 2: the zig-zag product. The second operation operation is called the *zig-zag product*. The goal of the zig-zag product is to reduce the degree d. The construction is much more subtle than graph powering.

Let $H = (V_0, E_0)$ be a constant degree expander with d vertices and degree d_0. The number of vertices in H is chosen to match the degree of G exactly. We identify the vertice of H with the set of indices $[d] = \{1, \ldots, d\}$.

The problem with G is that a random step takes $\log(d)$ bits, and $\log(d)$ is too large. We use H as a gadget to reduce this complexity. Since H is an expander, it rapidly approaches the stationary distribution, and the destination of single random step a uniform distribution over $V = [d]$. A random step in H takes only $\log(d_0) = O(1)$ random bits. This vertex in H maps back to a choice of neighbor in G.

Thus we use H to simulate the choices of a random walk in G. We maintain one vertex v_H in H and one vertex v_G in G. Rather than taking a random step in G (which requires $\log(d)$ bits), we take a random step on H (which requires only $\log(d_0)$ bits). The new index of v_H in H maps to a neighbor of v_G in G, and we move from v_G to this neighbor.

Intuitively, if H is an expander, then the random step in H should behave similarly to sampling a uniformly random vertex in H. In that case the choice of neighbor in G behaves similarly to a uniformly random choice of neighbor, and overall, v_G follows a path similar to a random walk. Meanwhile we only need $\log(d_0)$ bits to implement this "pseudo-random" step.

Now the action just described is not symmetric. More precisely, the arcs these steps describe on the product set $V_1 \times V_2$ do not given an undirected graph. To ensure we have an undirected graph we will have to append a few additional steps.

We are now prepared to define the zig-zag product formally. The *zig-zag product*, denoted $\mathcal{Z}(G \mid H)$, is a regular graph with vertex set $V_1 \times V_2$ and degree d_0^2. Each edge in $\mathcal{Z}(G \mid H)$ consists of a step in H (with d_0 degrees of freedom), a predetermined "zig-zag" step (with 0 degrees of freedom), followed by a step in H (with d_0 degrees of freedom). To define this formally, let $(v_1, i_1) \in V_1 \times V_2$ be a vertex. For $(k_1, k_2) \in [d_0] \times [d_0]$, the (k_1, k_2)th neighbor of (v_1, i_1) is the point (v_2, i_4) obtained by the following steps.

$$\begin{pmatrix} v_1 \\ i_1 \end{pmatrix} \xrightarrow{\text{(a)}} \begin{pmatrix} v_1 \\ i_2 \end{pmatrix} \xrightarrow{\text{(b)}} \begin{pmatrix} v_2 \\ i_2 \end{pmatrix} \xrightarrow{\text{(c)}} \begin{pmatrix} v_2 \\ i_3 \end{pmatrix} \xrightarrow{\text{(d)}} \begin{pmatrix} v_2 \\ i_4 \end{pmatrix}$$

where:

- (a) is a step in H from i_1 to its k_1th neighbor, i_2.
- (b) moves v_1 to the i_2th outgoing neighbor of v_1, v_2.
- (c) moves from i_2 to i_3 where v_1 is the i_3th neighbor of v_2.
- (d) moves i_3 to its k_2th neighbor in H.

The most important steps are the first two. This is where a random step in H induces a step in G. The remaining two steps makes the process reversible and the resulting graph an undirected graph. (Veryfing this is left to the reader.)

Consider an edge starting from (v_1, i_1) and ending at (v_2, i_4). v_1 and v_2 are neighbors in G. However, i_1 and i_4 are not (necessarily) neighbors! The discontinuity arises in the third step, where the relationship between v_1 and v_2 in G is used to "teleport" i_2 to i_3. Luckily, all we will really care about is preserving the connectivity in G. If $s, t \in V$ are connected in G, then for each index $i \in [d]$, there is an index $j \in [d]$ such that (s, i) and (t, j) are connected. Conversely, if (s, i) and (t, j) are connected in $\mathcal{Z}(G \mid H)$, then s and t are connected in G.

Analyzing the zig-zag product is the most involved part of the overall proof. We summarize its properties in the following lemma and defer the proof to later.

Lemma 23.4. *Let $G = (V, E)$ be a regular undirected graph with n vertices and degree d, with spectral gap γ_G. Let H be a regular undirected graph with d vertices and degree d_0. Then $\mathcal{Z}(G \mid H)$ is a regular undirected graph with nd vertices, degree d_0^2 and spectral gap $\gamma_G \gamma_H^2$.*

Combing powering with the zig-zag product. When we combine the zig-zag product with powering we get the following. For technical reasons it is convenient to assume G is regular with degree d^2 for some integer d.

Lemma 23.5. *Let G be a regular undirected graph with n vertices, degree d^2, and spectral gap γ_G. Let H be a regular undirected graph with d^4 vertices, degree d, and spectral gap γ_H. Then $\mathcal{Z}(G^2 \mid H)$ is a regular undirected graph with $d^4 n$ vertices, degree d^2, and spectral gap $\left(1 - (1-\gamma_G)^2\right)\gamma_H^2$.*

For (s,t)-connectivity, with some preprocessing, we can assume that G is a regular graph on n vertices with constant degree d^2 and spectral gap $\geq 1/\operatorname{poly}(n)$. We can also assume that there exists a regular expander H on d^4 vertices, degree d, and spectral gap $\geq 3/4$. If $\gamma_G \leq 1/16$, then by Lemma 23.5 $\mathcal{Z}(G^2 \mid H)$ has spectral gap

$$\geq 1.01\gamma_G.$$

That is, we increase the spectral gap by a constant factor! By repeating the construction $O(\log n)$ times, the final graph has constant expansion! The degree, meanwhile, is still d^2 – the same as before, and a constant.

We need to check that the number of vertices did not blow up too much. Each time we apply Lemma 23.5, the number of vertices increases by a constant factor d^4. Therefore, after $O(\log n)$ iterations the number of vertices increases by a $\operatorname{poly}(n)$-factor. Altogether we have a constant degree graph with $\operatorname{poly}(n)$ vertices and constant spectral-gap — primed for derandomizing the random walk approach.

To keep the space at $O(\log n)$ we do not explicitly construct the generated expander. Instead we will simulate walks on the expander implicitly.

All that said, we still need to verify that we can run simulate a random walk on the generated expander in $O(\log n)$ space. Let G_0 denote the input graph (with n vertices and constant degree d^2) and let $G_k = \mathcal{Z}\left(G_{k-1}^2 \,\middle|\, H\right)$ be the graph obtained after the kth iteration of Lemma 23.5. To simulate a random walk on one of these generated graphs, we need to be able to answer the following *ith-neighbor query*. This query takes as input a vertex v and an index i, and returns the ith neighbor of v. We need to show how to implement each query with small space.

We claim the following for each index j.

1. *The space required to execute an ith-neighbor query on G_j^2 is $O(1)$ plus the space required to simulate a step on G_j.*
2. *The space required to execute an ith-neighbor query on $\mathcal{Z}\left(G_j^2 \,\middle|\, H\right)$ is $O(1)$ plus the space required to simulate a step on G_j^2.*

If the above hold, then the space required to simulate a step on G_k is $O(k)$, as desired.

Consider the first claim, for G_j^2. We are given a vertex v_1 in G_j^2 and two indices $i_1, i_2 \in [d^2]$. We query (v, i_1) to take a step in G_j, which returns a vertex v_2 in G_j. We then query (v, i_2) to take a step in G_j which returns a vertex w_3. The maximum amount of space we ever use is $O(1)$ plus the space recursively required to take a step in G_j.

Consider the second claim, where we are simulating a step on $\mathcal{Z}\left(G_j^2 \middle| H\right)$. We are given a vertex (v_1, i_1), where v_1 is a vertex in G_j^2 and i_1 is a vertex in H (and at most a constant). We are also given two indices $j_1, j_2 \in [d_0]$ and want to return the (j_1, j_2)th neighbor of (v_1, i_1). We first take a step in H from i_1 to i_2, using $O(1)$ space. We then query G_j^2 for the i_2th edge from v_1 to some v_2 in G_j^2. We then query, for each $i_3 \in [d]$, the i_3th edge from v_2 in G_j^2 until we find that v_1 is the i_3th edge from v_2. Each of these queries take $O(1)$ space plus the space from the recursive call to G_j^2. Finally we use j_2 to update from i_3 to i_4 in H, in $O(1)$ space. The maximum amount of space we ever use is $O(1)$ plus the maximum amount of space recursively used when querying G_j^2.

All put together, for $k = O(\log(n))$, G_k is a constant degree expander with $\text{poly}(n)$ that preserves connectivity from G. Moreover we can navigate G_k implicitly via ith-neighbor queries in $O(\log n)$ space per query. We derandomize the random walk algorithm on G_k, which gives a deterministic algorithm for connectivity in G.

Up to proving Lemma 24.8 in the subsequent section, this completes the proof of Theorem 23.2.

23.3 Preliminaries

We introduce some mathematical background needed to analyze the zig-zag product.

23.3.1 Tensors of linear maps

Recall that $\mathbb{R}^{m\times n}$ denotes the space of linear maps from the vector space $\mathbb{R}^n$ to the vector space $\mathbb{R}^m$. $\mathbb{R}^{m\times n}$ itself is a real vector space with inner product given by

$$\langle A, B\rangle = \sum_{i,j} A_{ij}B_{ij} = \text{trace}\left(A^T B\right).$$

Two helpful identities are

$$\langle A, c \otimes d\rangle = \langle c, Ad\rangle$$
$$\langle (a \otimes b), (c \otimes d)\rangle = \langle a, c\rangle\langle b, d\rangle$$

for $A \in \mathbb{R}^{m \times n}$, $a, c \in \mathbb{R}^m$ and $b, d \in \mathbb{R}^n$, which the reader may verify.

For $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{n \times n}$,the *outer product* (a.k.a the *tensor product*) of A and B, denoted $A \otimes B$, is the linear map over $\mathbb{R}^{m \times n}$ defined by

$$\langle x, ((A \otimes B)C)y \rangle = \left\langle A^{\mathsf{T}} x, C B^{\mathsf{T}} y \right\rangle, \tag{23.1}$$

or equivalently,

$$(A \otimes B)C \overset{\text{def}}{=} A C B^{\mathsf{T}}, \tag{23.2}$$

for $C \in \mathbb{R}^{m \times n}$. We leave it to the reader to verify that this defines a linear map over $\mathbb{R}^{m \times n}$. We have the identities

$$(A \otimes B)(c \otimes d) = Ac \otimes Bd, \tag{23.3}$$
$$(A \otimes (B + C)) = (A \otimes B) + (B \otimes C), \tag{23.4}$$
$$(A \otimes B)^{\mathsf{T}} = \left(A^{\mathsf{T}} \otimes B^{\mathsf{T}}\right) \tag{23.5}$$

which are also left to the reader to verify. The reader may also verify that (23.3) implies Eqs. (23.1) and (23.2). The last identify implies that if A and B are symmetric, then so is $A \otimes B$.

Lemma 23.6. *$(A \otimes B)$ has eigenvalue-eigenvectors pairs of the form $(\lambda_A \lambda_B, x \otimes y)$, where x is an eigenvector of A with eigenvalue A and y is an eigenvector of B with eigenvalue λ_B.*

Proof. For each pair (λ_A, x) and (λ_B, y), we have

$$(A \otimes B)(x \otimes y) = Ax \otimes By = \lambda_A x \otimes \lambda_B y = \lambda_A \lambda_B (x \otimes y),$$

so $(x \otimes y)$ is an eigenvector of $(A \otimes B)$ with eigenvalue $\lambda_A \lambda_B$. There are m choices of (λ_A, x) and n choices of (λ_B, y) so together this gives mn eigenvalue/vector pairs. Since $(A \otimes B)$ acts on an $(m \times n)$-dimensional vector space, these are all the eigenvalue/vector pairs □

Tensor products of random walks. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs with random walk matrices R_1 and R_2. Consider the random walk on $V_1 \times V_2$ where given a pair of vertices $(v_1, v_2) \in V_1 \times V_2$, we take a random step from v_1 to w_1 according to R_1 and a random step from v_2 to w_2 according to R_2. The distribution of (w_1, w_2) is described by the $m \times n$ matrix

$$(R_1 1_{v_1}) \otimes (R_2 1_{v_2})$$

where 1_v denotes the $\{0,1\}$-indicator vector for a vertex v. More generally, given a distribution $P \in \mathbb{R}_{\geq 0}^{m \times n}$ over $V_1 \times V_2$, taking random steps along R_1 and R_2 simultaneously gives the distribution

$$(R_1 \otimes R_2)P = R_1^{\mathsf{T}} P R_2.$$

23.3.2 The operator norm

The *operator norm* of a linear map A is defined by

$$\|A\| = \sup\{\|Ax\| : \|x\| = 1\}.$$

$\|A\|^2$ is the maximum eigenvalue of the positive semi-definite matrix $A^{\mathsf{T}}A$. For a symmetric map A, this is the maximum absolute eigenvalue of A.

We have the identities

$$\|A + B\| \leq \|A\| + \|B\|, \tag{23.6}$$

$$\|AB\| \leq \|A\|\|B\|, \tag{23.7}$$

$$\|A \otimes B\| \leq \|A\|\|B\|, \tag{23.8}$$

the proofs of which are left to the reader.

23.4 Analysis of the zig-zag product

Lemma 24.8. *Let $G = (V, E)$ be a regular undirected graph with n vertices and degree d, with spectral gap γ_G. Let H be a regular undirected graph with d vertices and degree d_0. Then $\mathcal{Z}(G \mid H)$ is a regular undirected graph with nd vertices, degree d_0^2 and spectral gap $\gamma_G \gamma_H^2$.*

Let $R_{\mathcal{Z}}$ denote the random walk matrix of $\mathcal{Z}(G^2 \mid H)$. Let R_H be the random walk matrix of H. We can write a random step in the zig-zag graph as

$$R_{\mathcal{Z}} = (I \otimes R_H) Z (I \otimes R_H),$$

where $(I \otimes R_H)$ represents the action where we take a single random step in H but leave the G-coordinate fixed, and Z is the (deterministic) zig-zag step that updates the G-coordinate and transports the H-coordinate, as described above. Ultimately, we want to analyze the spectral gap of $(I \otimes R_H) Z (I \otimes R_H)$.

To give some intuition, recall that H is an expander. That is, taking a few steps in R_H is almost as random as sampling a uniformly random vertex from H. To formalize this connection, $S = (\mathbb{1} \otimes \mathbb{1})/d : \Delta^{V_H} \to \Delta^{V_H}$ be the "random walk" that samples a vertex from H uniformly at random. Intuitively, $R_H \approx S$. Suppose we substitute S for R_H in our expression for the random walk in the zig-zag product, giving,

$$(I \otimes S)Z(I \otimes S).$$

This step describes a zig-zag product of G with the graph H', which is a complete graph with a self-loop at every vertex. Let us walk through a random step in the zig-zag product $\mathcal{Z}(G \,|\, H')$.

1. Starting from $(v_1, i_1) \in G \times [d]$, we first take a random step in H' from i_1 to i_2. By definition of H', $i_2 \in [d]$ is selected uniformly at random.
2. We then move from v_1 to its i_2th neighbor v_2.
3. Then, we move from i_2 to i_3 where v_1 is the i_3th neighbor of v_1.
4. Then we take a step in H' move i_3 to a uniformly index $i_4 \in [d]$.

Overall, we move from (v_1, i_1) to (v_2, i_4) where v_2 is a uniformly random neighbor of v_1, and i_4 is a uniformly random vertex in H'. That is,

$$(I \otimes S)Z(I \otimes S) = (R_G \otimes S).$$

This is a much simpler step than the zig-zag product on an arbitrary graph, and can be analyzed directly. The second coordinate is essentially just uniformly random noise, and the first coordinate is walking in G. The second coordinate is mathematically irrelevant and the spectral gap of $\mathcal{Z}(G \,|\, H')$ is precisely γ_G, the spectral gap of G.

Of course, S is not exactly R_H, and additionally, the zig-zag product of G with H' does not decrease the degree as we would like. (In fact, it *increases* the degree.) It remains to quantify the difference between $(I \otimes R_H)Z(I \otimes R_H)$ and $(I \otimes S)Z(I \otimes S)$, which reflects the difference between R_H and S. As we will make more explicit below, the spectral gap of H, γ_H, is also a reflection of the difference between R_H and S. (S has spectral gap 1) This difference between R_h and S, and the correspondance between the difference and γ_H, is why the spectral gap decreases from γ_G to $\gamma_G \gamma_H^2$.

Previously when analyzing the spectral gap of undirected random walks, we applied the spectral theorem via similarity to the normalized random walk matrix. But if the graph is regular, then the random walk matrix is already symmetric, and in fact the same as the normalized random walk matrix. This applies to G, H, and $\mathcal{Z}(G \,|\, H)$.

$\mathcal{Z}(G \,|\, H)$ has stationary distribution $(\mathbb{1} \otimes \mathbb{1})/d^2$. Therefore $\mathbb{1} \otimes \mathbb{1}$ is the first eigenvector of $R_{\mathcal{Z}}$. To bound the spectral gap of $R_{\mathcal{Z}}$, we want to bound

$$|\langle x, R_{\mathcal{Z}} x \rangle|$$

over all x orthogonal to $\mathbb{1} \otimes \mathbb{1}$.

We can apply the spectral theorem to R_H by similarity to the normalized random walk matrix. Here, because H is a regular graph, $R_H = A/d$ is already a symmetric matrix (and coincides with the normalied random walk matrix) and we can apply the spectral theorem directly. By the spectral theorem for symmetric maps, combined the the Perron-Frobenius theorem for random walks, we have

$$R_H = u_1 \otimes u_1 + \lambda_2(u_2 \otimes u_2) + \cdots + \lambda_n(u_n \otimes u_n),$$

where $u_1, \dots, u_n \in \mathbb{R}^d$ forms an orthonormal bases and $\lambda_2, \dots, \lambda_n \in [1 - \gamma_H, \gamma_H - 1]$. Recall that because R_H is regular, the stationary distribution is the uniform distribution, $\mathbb{1}/d$. Since $R_H\mathbb{1} = \mathbb{1}$, the first eigenvector u_1 must be (proportional to) $\mathbb{1}$. This gives

$$\begin{aligned} R_H &= \frac{1}{d}(\mathbb{1} \otimes \mathbb{1}) + \lambda_2(u_2 \otimes u_2) + \cdots + \lambda_n(u_n \otimes u_n) \\ &= S + \lambda_2(u_2 \otimes u_2) + \cdots + \lambda_n(u_n \otimes u_n). \end{aligned}$$

Let $R_H' = R_H - \gamma_H S$; then R_H' has all its eigenvalues in the range $[\gamma_H - 1, 1 - \gamma_H]$.

We factor $R_{\mathcal{Z}}$ as

$$\begin{aligned} (I \otimes R_H)Z(I \otimes R_H) &= (I \otimes (\gamma_H S + R_H'))Z(I \otimes (\gamma_H S + R_H')) \\ &= (\gamma_H(I \otimes S) + (I \otimes R_H'))Z(\gamma_H(I \otimes S) + (I \otimes R_H')) \\ &= \gamma_H^2(I \otimes S)Z(I \otimes S) + \gamma_H(I \otimes S)Z(I \otimes R_H') \\ &\qquad + \gamma_H(I \otimes R_H')Z(I \otimes S) + (I \otimes R_H')Z(I \otimes R_H'). \end{aligned}$$

Let us first analyze the last 3 terms.

Let $x \in \mathbb{R}^{V \times d}$ be any unit vector orthogonal to the uniform distribution. We have

$$\begin{aligned} |\langle x, (I \otimes S)Z(I \otimes R_H')x\rangle| &= \|(I \otimes S)Z(I \otimes R_H')x\| \\ &\leq \|(I \otimes S)\|\|Z\|\|(I \otimes R_H')\| \leq 1 - \gamma_H. \end{aligned}$$

the third term contributes $1 - \gamma_H$ (times another γ_H) and the fourth term contributes $(1 - \gamma_H)^2$.

For the finale of our analysis, consider the remaining term, $(I \otimes S)Z(I \otimes S)$. As observed earlier, we have

$$(I \otimes S)Z(I \otimes S) = (R_G \otimes S) \qquad (!)$$

Note that $(R_G \otimes S)$ is the tensor product of G and H with has the same stationary distribution; namely the uniform distribution. Moreover, $(A \otimes S)$ has the same eigenvalues as A with the same multiplicity, since S has only one eigenvector with eigenvalue 1 and the rest are all 0. In particular, for any vector $x \in \mathbb{R}^{V \times d}$ orthogonal to the stationary distribution on $V \times [d]$, we have

$$|\langle x, (I \otimes S) Z (I \otimes S) x \rangle| = |\langle x, (R_G \otimes S) x \rangle| \leq 1 - \gamma_G.$$

Adding everything together we have

$$\gamma_H^2 (1 - \gamma_G) + 2\gamma_H (1 - \gamma_H) + (1 - \gamma_H)^2 = 1 - \gamma_G \gamma_H^2,$$

as desired.

23.5 Additional notes and materials

This topic is also covered in [HLW06, §9] and [Vad12, §4.4], which discuss additional related topics.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

23.6 Exercises

Exercise 23.1. Prove Lemma 23.3.

Exercise 23.2. Prove that the zig-zag product $\mathcal{Z}(G \,|\, H)$ is an undirected graph.

Exercise 23.3. Prove that the tensor product $A \otimes B$ defines a linear map over $\mathbb{R}^{m \times n}$ for $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{n \times n}$.

Exercise 23.4. Prove the tensor product identities in Eqs. (23.3)–(23.5).

Exercise 23.5. Prove the operator norm inequalities in Eqs. (23.6)–(23.8).

Exercise 23.6. The goal of this exercise is to develop a randomized construction of a constant degree expander.

Let $d \in \mathbb{N}$ be a parameter to be determined. Let n be even. Consider the random graph over n vertices where the edges are the disjoint union of (the edges of) d uniformly random perfect matchings over the vertices. (G can have parallel edges.) We will first show that G has conductance at least c for some constant $c > 0$. The connection to constant degree expanders is made at the end.

The key claim is as follows.

With nonzero probability, for all disjoint S and T such that $k = |S| \leq n/2$ and $|T| = k/6$, some vertex in S is matched to a vertex outside $S \cup T$.

1. Show that the claim implies that $|\delta(S)| \geq |S|/6$ for all S with $|S| \leq n/2$.
2. Show that if also $d = O(1)$, then the claim above implies that the union of matchings gives a graph with constant conductance.

We will prove this claim probabilistically: our first goal is to fix S and T, and bound the probability that all of S is matched to $S \cup T$. To this end, for fixed S and T, consider the following randomized procedure which produces a perfect matching.

```
1. Index the vertices $v_1, \dots, v_n$ such that $S = \{v_1, \dots, v_k\}$ and
   $T = \left\{v_{k+1}, \dots, v_{7k/6}\right\}$.
2. Repeat $n/2$ times:
   A. Let $v_i$ be the unmatched vertex of smallest index.
   B. Sample $v_j$ from the remaining unmatched vertices (excluding $v_i$) uniformly at random.
   C. Match $v_i$ with $v_j$.
```

3. Prove that the procedure above generates a uniformly random perfect matching.
4. Show that the probability that S is matched to $(S \cup T)$ in all d matchings is at most $\binom{n}{k}^{-\alpha d}$. for some constant $\alpha > 0$. (I got $\alpha = 1/6$.)[1]
5. Now prove the key claim above for some constant d.

[1]For a single matching, the probability that S is matched to $(S \cup T)$ is bounded above by the probability that the first $k/2$ choices of v_j's are in $S \cup T$.

Having now established that $O(1)$ random matchings have constant conductance with high probability:

6. Design and analyze an algorithm that produces a constant degree expander. (That is, a regular constant degree graph whose random walk has constant spectral gap γ.)[2]

[2] *Hint:* don't forget about the minimum eigenvalue.

Chapter 24

Reducing randomness with random walks

24.1 Introduction

Recall that a language L is in the *class* $\boldsymbol{P}$ if there is a deterministic poly(n)-time algorithm that decides if an input x of size n is in L. This notion extends to randomized algorithms as follows.

Definition 24.1. *A language L is in the* class **RP** *if there is a randomized polynomial time algorithm deciding L with the following probabilistic "one-sided" error guarantee:*

1. *Given an input $x \in L$, the algorithm decides that $x \in L$ with constant probability (say, $1/2$).*

2. *Given an input $x \notin L$, the algorithm always decides that $x \notin L$.*

A language L is in the class **BPP** *if there is a randomized polynomial time algorithm deciding L with the following probabilistic "two-sided" error guarantee:*

1. *Given an input $x \in L$, the algorithm decides that $x \in L$ with probability* $2/3$.

2. *Given an input $x \notin L$, the algorithm decides that $x \notin L$ with probability* $2/3$.

We have the subsets

$$\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{BPP},$$

since of course, no error (**P**) is better than one-sided error (**RP**), which is better than two-sided error (**BPP**). It is a major open question if these are equal. Many believe that **P** = **BPP**. In practice, researchers treat a randomized algorithm with two-sided error as a strong indicator for the existence of a deterministic one. Still, we do not know really if **P** equals **RP** or if **RP** equals **BPP**.

There is theoretical interest, sometimes under the heading of *pseudorandomness*, in a refined understanding of how much randomness is required for various problems. While the holy grail, **P** vs **BPP**, is hard to attack, there is a rich body of literature and results moving towards a conclusion, producing many algorithmic ideas of independent interest along the way.

Let $L \in \mathbf{RP}$, and fix an input size n. Suppose that an algorithm for L requires m random bits to decide L with one-sided error $1/2$. If we want to decrease the error to δ, for some δ, then we could independently repeat the algorithm $\lceil \log 1/\delta \rceil$ times, taking the disjunction of responses. This takes $O(m \log(1/\delta))$ random bits total. While we typically de-emphasize the logarithmic overhead incurred from repetition, it is a deep and natural question to ask if one can reduce or avoid the logarithmic overhead. Surprisingly, one can:

Theorem 24.2. *Given an algorithm in* ***RP*** *that uses* m *random bits and has probability of error at most* $1/2$*, one can decrease the error probability to* δ *while increasing the running time by a factor of* $\log(1/\delta)$ *and using a* total *of* $m + O(\log(1/\delta))$ *random bits.*

There is an analogous result for **BPP**. The algorithm is the same as for **RP**, while the analysis is different.

Theorem 24.3. *Given an algorithm in* ***BPP*** *that uses* m *random bits to achieve error* $1/3$ *(on both sides), one can decrease the error probability to* δ *while increasing the running time by a factor of* $\log(1/\delta)$ *and using a* total *of* $m + O(\log(1/\delta))$ *random bits.*

The rest of this discussion is about proving Theorems 24.2 and 24.3.

24.2 High level overview: amplification by random walks

The algorithm that obtains the better-than-repetition amplification above is conceptually very clean. Let G be a constant degree expander with vertex set $V = \{0,1\}^m$ — that is, a vertex for every possible bit string of m bits. (We will have to address how to implicitly build such a G later, but for now let us assume G is given.) First, select a uniformly random vertex $v_0 \in V$. (This takes m random bits). Then take a random walk in G for $O(\log(1/\delta))$ steps. Each step takes $O(1)$ random bits. For each vertex v_i along the walk, use v_i as the input for a new instance of the algorithm. For algorithms in **RP**, output the disjunction (the "or") of all the outputs. For algorithms in **BPP**, output the majority vote.

It is easy to see that we use $m + O(\log(1/\delta))$ random bits in total. To complete the proofs of Theorems 24.2 and 24.3 there are two basic issues to address.

1. We need to show that the bit strings generated by the expander are (probably) useful, for both the **RP** and **BPP** settings.

2. We need to show how to efficiently make such a graph G.

For the second point, since G is exponential-size, this construction has to be implicit.

Random walks for RP. Consider the first point — why bit strings generated by an expander graph act like totally random bit strings — for **RP**. The key lemma is as follows.

Lemma 24.4. *Let $G = (V, E)$ be a regular undirected graph whose random walk has spectral gap γ. Consider a t-step random walk $v_1, v_2, \ldots, v_t \in V$ where $v_1 \in V$ is chosen uniformly at random. For any set $B \subset V$, the probability that the entire random walk stays in B is*

$$(\mu + (1-\gamma)(1-\mu))^t,$$

where $\mu = |B|/|V|$.

We will prove this lemma later in Section 24.3. First let us derive Theorem 24.2 (on amplifying **RP** via expanders) in light of Lemma 24.4.

Let B be the set of vertices corresponding to "bad" bit strings causing the randomized algorithm to err. Amplifying the original algorithm by a constant number of repetitions as needed, we can make μ arbitrarily small; say, 1/2. By Lemma 24.4, if we take a random walk on an expander of bit string, the probability that the random walk stays in B — and all bit-strings are bad — drops at a rate of $(1-\gamma/2)^t$.

Random walks for BPP. We now present a similar lemma that is important for **BPP**.

Lemma 24.5. *Let $\epsilon \in (0, 1)$ be fixed. Let $G = (V, E)$ be a regular undirected graph whose random walk has spectral gap $\gamma \geq 1 - \epsilon$. Let $f : V \to [0, 1]$ be a fixed function of the vertices. Let*

$$\mu = \mathbf{E}[f(v)] \text{ where } v \in V \text{ is sampled uniformly at random.}$$

Consider a random walk $v_1, v_2, \ldots, v_k \in V$ where $v_1 \in V$ is chosen uniformly at random. Then for all $\beta > 0$,

$$\mathbf{P}\left[\left|\frac{1}{k}\sum_{i=1}^{k} f(v_i) - \mu\right| \geq \epsilon\mu + \beta\right] \leq ce^{\epsilon k((1+\epsilon)\epsilon - \beta)}$$

for a universal constant $c > 0$. In particular, for (say) $\epsilon \leq \mu/4$, we have

$$\mathbf{P}\left[\left|\frac{1}{k}\sum_{i=1}^{k} f(v_i) - \mu\right| \geq \epsilon\mu\right] \leq ce^{-\frac{\epsilon k \mu}{c}}$$

for a universal constant $c > 0$.

Analogous to our discussion for RP, Lemma 24.5 implies Theorem 24.3. We prove Lemma 24.5 in Section 24.4.

Efficiently and implicitly constructing the expander graph. To *make* the expander G, we apply the following theorem, which is a tweak on our previous construction for derandomizing random walks. We prove the following in Section 24.5.

Lemma 24.6. *Let $d \in \mathbb{N}$, and Let H be a regular undirected graph with d^8 vertices, degree d, and spectral gap $\geq 7/8$. Define graphs $G_1, G_2, \ldots$ by*

$$\begin{aligned} G_1 &= H^2 \\ G_{t+1} &= \mathcal{Z}\Big((G_t \otimes G_t)^2 \,\Big|\, H\Big) \end{aligned}$$

Then G_t has spectral gap $1/2$ and $2^{\Omega\left(2^t\right)}$ vertices. Simulating one step of a random walk in G_t takes $2^{O(t)}$ time and $O(\log(d))$ random bits.

Taking $t = \log\log m + O(1)$ gives the desired expander over $\{0,1\}^m$.

24.3 Amplifying RP: proof of Lemma 24.4

Let R be the random walk matrix on G. Let $P : \mathbb{R}^V \to \mathbb{R}^V$ be the projection onto $\mathbb{R}^B$; that is,

$$(Px)_v = \begin{cases} x_v & \text{if } v \in B \\ 0 & \text{otherwise.} \end{cases}$$

P is a linear function with $P = P^T$ and $P^2 = P$. One can think of P as the identity matrix restricted to $\mathbb{R}^B$ (and 0 everywhere else).

Consider the product PRP. Given a nonnegative vector x, $PRPx$ drops all the mass outside of B, take a step according to R, and again drops all of the mass outside of B. In particular, the probability that the entire walk stays in B is

$$\Big\langle \mathbb{1}, (PRP)^t(\mathbb{1}/n)\Big\rangle.$$

We claim that PRP has maximum eigenvalue $\leq 1-(1-\mu)\gamma$. If so, then

$$\left\langle \mathbb{1}, (PRP)^t(\mathbb{1}/n) \right\rangle = \frac{\left\langle \mathbb{1}, (PRP)^t \mathbb{1} \right\rangle}{\langle \mathbb{1}, \mathbb{1} \rangle} \leq (\gamma\mu + 1 - \gamma)^t,$$

which completes the proof.

Since G is regular, R is symmetric, and we can write

$$R = \frac{\gamma}{n}(\mathbb{1} \otimes \mathbb{1}) + R'$$

where R' has all of its eigenvalues in the range $[1-\gamma, \gamma-1]$. Then

$$PRP = \frac{\gamma}{n} P(\mathbb{1} \otimes \mathbb{1})P + PR'P.$$

We leave it to the reader to show that $PR'P$ has all its eigenvalues in the range $[1-\gamma, \gamma-1]$.

Consider the first term $(\gamma/n)P(\mathbb{1} \otimes \mathbb{1})P$. We claim that it has maximum eigenvalue $\gamma\mu = |B|/n$, which would give the overall bound of

$$\|PRP\| = \left\| \frac{\gamma}{n} P(\mathbb{1} \otimes \mathbb{1})P \right\| + \|PR'P\| \leq \gamma\mu + 1 - \gamma,$$

as desired, where $\|\cdots\|$ is the operator norm.

We have

$$\frac{\gamma}{n} P(\mathbb{1} \otimes \mathbb{1})P = \frac{\gamma}{n}(P\mathbb{1} \otimes P\mathbb{1}).$$

The maximum eigenvector of the outer product $(P\mathbb{1} \otimes P\mathbb{1})$ is (proportional to) $P\mathbb{1}$, with eigenvalue

$$\frac{\langle P\mathbb{1}, P\mathbb{1} \rangle^2}{\langle P\mathbb{1}, P\mathbb{1} \rangle} = \langle P\mathbb{1}, P\mathbb{1} \rangle = |B|.$$

Thus

$$\left\| \frac{\gamma}{n}(P(\mathbb{1} \otimes \mathbb{1})P) \right\| = \frac{\gamma|B|}{n} = \mu.$$

This gives the desired bound.

24.4 Efficiently amplifying BPP

Lemma 24.5. *Let $\epsilon \in (0,1)$ be fixed. Let $G = (V, E)$ be a regular undirected graph whose random walk has spectral gap $\gamma \geq 1 - \epsilon$. Let $f : V \to [0,1]$ be a fixed function of the vertices. Let*

$$\mu = \mathbf{E}[f(v)] \text{ where } v \in V \text{ is sampled uniformly at random.}$$

Consider a random walk $v_1, v_2, \dots, v_k \in V$ where $v_1 \in V$ is chosen uniformly at random. Then for all $\beta > 0$,

$$\mathbf{P}\left[\left|\frac{1}{k}\sum_{i=1}^{k} f(v_i) - \mu\right| \geq \epsilon\mu + \beta\right] \leq c e^{\epsilon k((1+\epsilon)\epsilon - \beta)}$$

for a universal constant $c > 0$. In particular, for (say) $\epsilon \leq \mu/4$, we have

$$\mathbf{P}\left[\left|\frac{1}{k}\sum_{i=1}^{k} f(v_i) - \mu\right| \geq \epsilon\mu\right] \leq c e^{-\frac{\epsilon k \mu}{c}}$$

for a universal constant $c > 0$.

Proof. Initially the proof proceeds similarly to the Chernoff bound. We let us prove the inequality on the upper tail. The lower tail follows similarly, and then we can take the union bound over both. We have

$$\mathbf{P}\left[\sum_{i=1}^{k} f(v_i) \geq k(1+\epsilon)\mu + \beta\right] \leq \mathbf{E}\left[e^{\epsilon(f(v_1)+\cdots+f(v_k))}\right] e^{-\epsilon(1+\epsilon)k\mu - \epsilon\beta}. \tag{24.1}$$

The key identity is

$$\mathbf{E}\left[e^{\epsilon(f(v_1)+\cdots+f(v_k))}\right] = \frac{1}{n}\left\langle \mathbb{1}, F(RF)^k \mathbb{1}\right\rangle \text{ where } F = \operatorname{diag}\left(e^{\epsilon f(v_1)}, \dots, e^{\epsilon f(v_n)}\right)$$

is the diagonal matrix with the exponentiated values along the diagonal. One way to interpret the above is to first recall that R^k models k steps of a random walk. Then inserting F in between the R's is like collecting the values $e^{\epsilon f(v)}$ along the walk. We claim the following.

Claim 1. $\frac{1}{n}\left\langle F^{1/2}\mathbb{1}, F(RF)^k F^{1/2}\mathbb{1}\right\rangle \leq e^{\epsilon + (k(1+\epsilon)\epsilon(\mu+\epsilon))}$

Assuming claim 1 holds, we have

$$(24.1) \leq e^{\epsilon} \cdot e^{\epsilon k((1+\epsilon)\epsilon - \beta)},$$

as desired.

Let us now prove Claim 1. Since R has spectral gap $\geq 1-\epsilon$, and R is symmetric, we can pull out a $(1-\epsilon)$-fraction of its leading eigenvector (corresponding to the uniform distribution), writing

$$R = \frac{1-\epsilon}{n}(\mathbb{1} \otimes \mathbb{1}) + R'$$

where R' has eigenvalues between ϵ and $-\epsilon$. Thereby

$$F^{1/2} R F^{1/2} = \frac{1-\epsilon}{n} F^{1/2}(\mathbb{1} \otimes \mathbb{1}) F^{1/2} + F^{1/2} R' F^{1/2}.$$

We claim the following.

Claim 2. *$F^{1/2} R' F^{1/2}$ has its eigenvalues between $[\epsilon e^{\epsilon}, -\epsilon e^{\epsilon}]$.*

Claim 3. *$F^{1/2}(\mathbb{1} \otimes \mathbb{1}) F^{1/2}$ has maximum eigenvalue $\mathbf{E}\left[e^{\epsilon f(v)}\right]$.*

For the first claim regarding $F^{1/2} R' F^{1/2}$, for any vector x

$$\begin{aligned} \left\langle x, F^{1/2} R' F^{1/2} x \right\rangle &= \left\langle F^{1/2} x, R'\left(F^{1/2} x\right) \right\rangle \leq \epsilon \left\| F^{1/2} x \right\|^2 \\ &= \epsilon \langle x, F x \rangle \leq \epsilon e^{t} \|x\|^2, \end{aligned}$$

where we repeatedly invoke the fact that the maximum eigenvalue of a symmetric matrix is given by the Rayleigh quotient.

For the second claim, we have

$$F^{1/2}(\mathbb{1} \otimes \mathbb{1}) F^{1/2} = \left(F^{1/2}\mathbb{1} \otimes F^{1/2}\mathbb{1}\right)$$

which has maximum eigenvalue

$$\left\| F^{1/2}\mathbb{1} \right\|^2 = \langle \mathbb{1}, F\mathbb{1} \rangle = \sum_{v} e^{\epsilon f(v)} = n\, \mathbf{E}\left[e^{\epsilon f(v)}\right].$$

This establishes the second claim.

Combining the two claims above, we have that $F^{1/2} R F^{1/2}$ has maximum (absolute) eigenvalue

$$\left\| F^{1/2} R F^{1/2} \right\| \leq \epsilon e^{\epsilon} + (1-\epsilon)\, \mathbf{E}\left[e^{\epsilon f(v)}\right].$$

We expand the right hand side by the inequality $e^{\epsilon} \leq 1 + \epsilon + \epsilon^2$ for small $|t|$, giving the upper bound

$$\begin{aligned}\left\|F^{1/2}RF^{1/2}\right\| &\leq \epsilon\Big(1 + \epsilon + \epsilon^2\Big) + (1-\epsilon)\,\mathbf{E}\Big[1 + \epsilon f(v) + \epsilon^2 f(v)\Big] \\ &= \epsilon\Big(1 + \epsilon + \epsilon^2\Big) + (1-\epsilon)\Big(1 + \epsilon\mu + \epsilon^2\mu\Big) \\ &\leq 1 + \Big(\epsilon + \epsilon^2\Big)(\mu + \epsilon) \\ &\leq e^{\left(\epsilon+\epsilon^2\right)(\mu+\epsilon)}\end{aligned}$$

In turn, for the kth power, we have

$$\left\|\Big(F^{1/2}RF^{1/2}\Big)^k\right\| = \left\|F^{1/2}RF^{1/2}\right\|^k \leq e^{k(1+\epsilon)\epsilon(\mu+\epsilon)}.$$

Finally, returning to the original quantity we wanted to sum, we have

$$\frac{1}{n}\Big\langle F^{1/2}\mathbb{1}, F(RF)^k F^{1/2}\mathbb{1}\Big\rangle \leq e^{k(1+t)\epsilon(\mu+\epsilon)}\,\mathbf{E}\Big[e^{\epsilon f(v)}\Big] \leq (1 + O(\epsilon))e^{(k(1+\epsilon)\epsilon(\mu+\epsilon))}.$$

as desired for Claim 1. □

24.5 Efficiently making large expanders

It remains to be shown that large expanders can be constructed efficiently. Previously, in the interest of deterministic connectivity, we studied the amplification

$$G \mapsto \mathcal{Z}\Big(G^2 \,\Big|\, H\Big),$$

where the degrees and sizes of G and H are appropriately set. The primary goal of the goal of that exercise was to increase the spectral gap given an input graph (with bad spectral gap) in a space efficient manner.

Here our goal is slightly different, because simply want to make an expander over 2^m vertices without the burden of some bad input graph. That is, we simply want to make a large - very, very large - expander. Note that we want to make this graph implicitly and be able to take a step in the graph in $O(\text{polylog}(m))$ time per step – importantly, this is *doubly* logarithmic in the number of vertices, 2^m. If we apply the construction form connectivity starting from a constant sized expander, we will end up needing $O(m)$ iterations to get up to 2^m vertices, since each iteration increases the number of vertices of a constant factor. Thus, in contrast to before, the goal is to *increase the number of vertices given an expander* as efficiently as possible.

Let us now restate the main lemma that we need to prove.

Lemma 24.6. *Let $d \in \mathbb{N}$, and Let H be a regular undirected graph with d^8 vertices, degree d, and spectral gap $\geq 7/8$. Define graphs $G_1, G_2, \dots$ by*

$$\begin{aligned} G_1 &= H^2 \\ G_{t+1} &= \mathcal{Z}\Big((G_t \otimes G_t)^2 \,\Big|\, H\Big) \end{aligned}$$

Then G_t has spectral gap $1/2$ and $2^{\Omega(2^t)}$ vertices. Simulating one step of a random walk in G_t takes $2^{O(t)}$ time and $O(\log(d))$ random bits.

We first recall the first two lemma's that we proved previously.

Lemma 24.7. *Let $G = (V, E)$ be a regular undirected graph n vertices and degree d. Then G^k is a regular undirected graph on V with degree d^k, with random walk map R^k. If R has spectral gap γ, then R^k has spectral gap $1 - (1-\gamma)^k$.*

Lemma 24.8. *Let $G = (V, E)$ be a regular undirected graph with n vertices and degree d, with spectral gap γ_G. Let H be a regular undirected graph with d vertices and degree d_0. Then $\mathcal{Z}(G \,|\, H)$ is a regular undirected graph with nd vertices, degree d_0^2 and spectral gap $\gamma_G \gamma_H^2$.*

The second lemma, regarding the zig-zag product, required the following structural lemma about the tensor product of undirected graphs and their random walks.

Lemma 24.9. *Let G_1 and G_2 be regular undirected graphs with degrees d_1 and d_2 and random walk matrices R_1 and R_2 respectively. Then $G_1 \otimes G_2$ is a regular undirected graph with degree $d_1 d_2$, Then the random walk matrix of $G_1 \otimes G_2$, denoted $R_1 \otimes R_2 : \mathbb{R}^{V_1 \times V_2} \to \mathbb{R}^{V_1 \times V_2}$, is also symmetric. The map*

$$(v_1, v_2) \in \mathbb{R}^{V_1} \times \mathbb{R}^{V_2} \mapsto v_1 \otimes v_2 \in \mathbb{R}^{V_1 \times V_2}$$

gives a one-to-one correspondance between pairs of eigenvectors from G_1 to G_2, where an eigenvector v_1 with eigenvalue λ_1 of G_1 and an eigenvector v_2 with eigenvalue λ_2 of G_2 maps to an eigenvector $v_1 \otimes v_2$ of $G_1 \otimes G_2$ with eigenvalue $\lambda_1 \lambda_2$.

Let $d \in \mathbb{N}$ be a fixed constant and let H be an undirected regular graph d^4 vertices, degree d, and spectral gap $7/8$. Let $G_0 = H^2$. We now generate graphs $G_1, G_2, \dots$ iteratively by

$$G_{i+1} = \mathcal{Z}\Big((G_i \otimes G_i)^2 \,\Big|\, H\Big).$$

The various parameters of interest develop as follows.

Graph	G	$\to$	$G \otimes G$	$\to$	$(G \otimes G)^2$	$\to$	$\mathcal{Z}\Big((G \otimes G)^2 \,\Big\vert\, H\Big)$
Vertices	n	$\to$	n^2	$\to$	n^2	$\to$	$n^2 d^4$
Degree	d^2	$\to$	d^4	$\to$	d^8	$\to$	d^2
$\boldsymbol{\gamma}$	γ	$\to$	γ	$\to$	$2\gamma - \gamma^2$	$\to$	$(2\gamma - \gamma^2)(7/8)^2$

We note that $\gamma \geq 1/2$ implies $(2\gamma - \gamma^2) \geq 1/2$, so the spectral gap never drops below $1/2$.

24.6 Additional notes and materials

See [Vad12] for additional topics in pseudorandomness.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

24.7 Exercises

Exercise 24.1. The definition of RP allows for error probability $1/2$ while BPP requires a constantly strictly less than $1/2$ (e.g., $1/3$). Here we explore why.

Give a concise description of the family of all languages L for which there is a randomized polynomial time algorithm that, given input x:

1. If $x \in L$, decides that $x \in L$ with probability at least $1/2$.
2. If $x \notin L$, decides that $x \notin L$ with probability at least $1/2$.

Exercise 24.2. Extend Lemma 24.4 to graphs G that are not regular.

Exercise 24.3. Extend Lemma 24.5 to graphs G that are not regular.

Chapter 25

Randomized Proofs and Verification by Random Walks

25.1 Randomized Proofs

Recall that the class **P** is the class of all polynomial time solvable problems, and **NP** is the class of all languages that can be decided in non-deterministic polynomial time. Equivalently, a language L is in **NP** if a membership $x \in L$ can be *proven* in polynomial time. This means there exists a (deterministic) polynomial time algorithm, called the *verifier*, that takes as input $x \in \{0,1\}^n$ and an additional polynomial-sized input $y \in \{0,1\}^{\mathrm{poly}(n)}$, called the *proof*. Based on x and y, the verifier decides if $x \in L$ according to the following protocol:

- If $x \in L$, then the verifier accepts x for some y.
- If $x \notin L$, then the verifier rejects x for all y.

Obviously $\mathbf{P} \subseteq \mathbf{NP}$, and an outstanding open problem is whether **P** is equal to **NP**. A long track record of failing to solve many **NP** problems of practical interest, combined with the equivalence class of **NP**-complete problems, suggests that they are not equal, but again, we cannot prove it. **P** vs **NP** is a fascinating question extending beyond computation; see for example [Wig09].

Consider these questions from a randomized point of view. Suppose we granted the verifier access to randomization, and relaxed our protocol to allow one-sided error. Consider the following protocol for input x and proof y:

- If $x \in L$, then for some y, the verifier always accepts x.
- If $x \notin L$, then for all y, the verifier accepts x with probability $\leq 1/2$.

The proof is accessed in an oracle model, where the verifier can query the ith bit from an oracle. For a language L, a *(nonadaptive) probabilistically checkable proof with r random bits and q queries*, denoted $\mathbf{PCP}(r, q)$ is one that, given an input x and oracle access to a proof y, decides if $x \in L$ via the following steps:

1. Given full access to x and $O(r)$ random bits, the verifier chooses $O(q)$ locations in y to query.

2. The verifier makes these $O(q)$ queries.
3. Based on x, the $O(r)$ random bits from (1), and the $O(q)$ queries in (2), the verifier decides if $x \in L$.

The verifier can spend polynomial time inspecting the input x and the outcomes of the $O(r)$ coin flips to choose its queries. The queries to the proof are nonadaptive — the ith query does not depend on the outcome of the previous $i - 1$ queries.

Of course any language on **NP** has such a verifier; i.e., $\mathbf{NP} \subseteq \mathbf{PCP}(0, \text{poly}(n))$. The question is, with randomization in hand, can the number of queries q be reduced from $\text{poly}(n)$ to $n^{o(1)}$?

Theorem 25.1 (PCP theorem)**.** *$\mathbf{NP} = \mathbf{PCP}(O(\log n), O(1))$. Every **NP** language has a probabilistically checkable proof with $O(\log n)$ random bits and $O(1)$ queries to the proof.*

The PCP theorem is philosphically very interesting, giving a robust, "error-correcting" extension to our deterministic notion of proofs.[1]

The PCP theorem has also had a large impact in *hardness of approximation.* As we know all too well, many problems are **NP**-hard, which motivates approximation algorithms for these problems. Are there also limits to approximations? Can we expect better and better approximations over time for all problems, or is there also a hard limit to approximations?

As a concrete example, consider the 3-SAT problem. The input is a SAT formula with exactly 3 literals per clause, and (in the optimization formulation) the goal is to satisfy as many clauses as possible. As discussed previously, if we randomly assign each variable, then we get a 7/8th approximation. Surely, such a simple and essentially oblivious algorithm could not be very good. Yet the PCP theorem leads to the following remarkable theorem that 7/8th is optimal:

Theorem 25.2 ([Hås01])**.** *For all $\epsilon > 0$, it is NP-Hard to obtain a $(7/8 + \epsilon)$-approximation to 3-SAT.*

There are hardness of approximation results for many problems, connected to one another via *approximation-preserving reductions.*

One direction of the PCP theorem is easier to prove than the other and left as an exercise to the reader.

Exercise 25.1. Prove that $\mathbf{PCP}(O(\log n), O(1)) \subseteq \mathbf{NP}$.

[1] The old joke is that the PCP theorem implies a much faster way to grade algorithms homework.

It remains to show $\mathbf{NP} \subseteq \mathbf{PCP}(O(\log n), O(1))$: that is, every language in **NP** has a probabilistically checkable proof.

The celebrated PCP theorem was developed in the late 80's and early 90's and was born out of earlier developments investigating *interactive proofs*. Unfortunately the mathematical techniques used in this original line of work have not been developed in this class. Instead will present a more recent proof by Dinur [Din07] that is considered to be simpler then the original proof of the PCP theorem. The machinery driving Dinur's approach will relate to recent discussions on random walks.

25.2 Constraint satisfaction problems

Let V be a set of n variables, and let A be a finite alphabet. A *k-ary constraint* consists of k variables $v_1, \dots, v_k \in V$ and a subset $S \subseteq A^k$. An assignment $\sigma : V \to A$ satisfies the constraint if $(\sigma(v_1), \dots, \sigma(v_k)) \in S$. In *$q$-ary constraint satisfaction problems*, we are given m q-ary clauses over a set of n variables V and a finite alphabet A. The goal is to maximize as many clauses as possible. For a CSP P, we let $\textsc{unsat}(P)$ denote the minimum fraction of unsatisfied clauses of P over all P. (e.g., $\textsc{unsat}(P) = 0$ iff P is satisfiable.)

Our discussion is about proving the following theorem in particular.

Theorem 25.3. *There are integers $q > 1, |A| > 1$ such that, given a q-ary CSP over alphabet A, it is NP-hard to decide whether*

1. *All clauses can be satisfied ($\textsc{unsat}(P) = 0$).*
2. *Less than or equal to half the clauses can be satisfied $\textsc{unsat}(P) \geq 1/2$.*

It is equivalent to the PCP theorem.

Theorem 25.4. *Theorems 25.1 and 25.3 are equivalent.*

The proof is left to the reader in the following exercise.

Exercise 25.2. Prove Theorem 25.1 and Theorem 25.3 are equivalent. Below we give part of the proofs, in both directions, to get you started.

1. *Theorem 25.1 $\Longrightarrow$ Theorem 25.3.* Suppose the PCP theorem, Theorem 25.1, is true. That is, every NP language L has a verifier on input x and proof y that reads $r = c \log n$ random bits and querys $q = O(1)$ bits from y, and correctly. We want to show that (1/2)-approximate for CSP — that is, deciding between

whether a CSP is (perfectly) satisfiable or if at most $1/2$ of the clauses can be satisfied — is NP-Hard.

Fix a language L in NP. Given input x of size n, we want to form a CSP problem P such that deciding between $\textsc{unsat}(P) = 0$ and $\textsc{unsat}(P) \geq 1/2$ is equivalent to deciding if $x \in L$. By the PCP theorem, there exists a verifier that flips at most $r = c\log(n)$ coins and reads $q = O(1)$ bits from the proof and decides whether to accept or reject. Let $A = \{0, 1\}$ be the alphabet, and make a boolean variable v_i for every location i of the proof that might be accessed by the randomized verifier. Note that this creates at most $q2^r = \text{poly}(n)$ boolean variables. Now, for each $z \in \{0, 1\}^r$, representing an outcome of the coin tosses, we defined a clause C_z with variables... and accepting the set of assignments...

2. *Theorem 25.3* $\implies$ *Theorem 25.1*. Conversely, suppose that it is NP-Hard to decide between $\textsc{unsat}(P) = 0$ and $\textsc{unsat}(P) \geq 1/2$ for a given CSP problem P. This means that for every language L, there is a transformation that, given an input x of size n, produces a q-ary CSP P_x with $\text{poly}(n)$ constraints such that $x \in L$ iff $\textsc{unsat}(P_x) = 0$ and $x \notin L$ iff $\textsc{unsat}(P_x) \geq 1/2$. We create a probabilisticaly checkable proof system where...

25.3 Graph CSP, and amplification

Consider the special case of binary CSP (i.e., $q = 2$). This means that every clause consists of two variables v_1, v_2 and a subset of satisfying pairs $S \subset A^2$. We will prove that binary CSP with a constant alphabet size (for some constant) is hard to approximate in the sense of Theorem 25.3.

Binary CSP can be modeled as a graph problem. We think of each variable $v \in V$ as a vertex. For every clause over two variables v_1, v_2, we have an edge between v_1 and v_2 labeled by that clause. Note that we can have parallel edges if there are multiple clauses for the same pair of vertices. To emphasize this graphical viewpoint, we call binary CSP problems *graph CSP* from now on.

Graph CSP is NP-Hard even for 3 letters in the alphabet. In particular, 3-colorability[2] is a special case of graph CSP that is NP-Hard. In our CSP terminology, this means it is NP-Hard to decide if $\textsc{unsat}(G) = 0$. Since 3-colorability has one constraint per edge we can recast this as saying that it is NP-Hard to decide if $\textsc{unsat}(G) = 0$ or if $\textsc{unsat}(G) \geq 1/m$, where m is the number of edges. We take this as our starting point, and the goal is now to "amplify" the graph CSP problem so

[2]3-Colorability asks if the vertices of a given graph can be labeled by one of three "colors" such that each color forms an independent set.

that it is NP-Hard to distinguish between $\textsc{unsat}(G) = 0$ or $\textsc{unsat}(G) \geq c$, for some fixed constant c.

Iterative amplification - an overview. For inspiration, we briefly recall the deterministic logspace algorithm for (s, t)-connectivity [Rei08]. This problem was actually easy for constant degree expanders, but of course the input graph is generally not a constant degree expander. The goal becomes to implicitly convert the graph into a constant degree expander. There we iterated between powering the graph — amplifying the spectral gap — and taking a zig-zag product with a constant degree expander — sparsifying the graph. Together these operations formed a net gain in the spectral gap, while keeping the degree constant. A logarithmic number of iterations transformed the input graph (implicitly) into a constant degree expander.

The proof of Dinur [Din07] is similar in spirit, trying to amplify a graph CSP by a series of transformations. In fact Dinur mentions deterministic log-space connectivity as an inspiration. Here we have three basic transformations:

1. *Expander-ification*, where we make the underlying graph a constant degree expander.
2. *Error amplification by powering*, where we amplify the $\textsc{unsat}$ of the graph by taking a power of the graph and creating new constraints appropriately.
3. *Alphabet reduction.* Where we reduce the alphabet size of the graph CSP (which increases in the powering step).

Each of these three steps take as input one graph CSP G and outputs another, G'. As we analyze these steps, we will be careful to ensure the following critical properties.

1. *Completeness:* If $\textsc{unsat}(G) = 0$, then $\textsc{unsat}(G') = 0$.
2. *Soundness:* $\textsc{unsat}(G') \geq \beta\, \textsc{unsat}(G)$ for some value $\beta > 0$.

For soundness, we want $\beta > 1$, but an individual operation may have $\beta < 1$ if it is principally concerned with managing other parameters (such as the alphabet size, or the degree). We will show that the three steps together gives $\beta > 1$, for a fixed constant β, while keeping the other parameters under control (and more precisely, universal constants).

Expander-ification. Let us break down these steps in a little more detail. The first step, "expander-ification", preprocesses G so that it is a constant degree expander. The constant degree is important because the subsequent power step has to blow up the size of the alphabet exponentially in the degree. The spectral gap is important for the alphabet reduction step after that.

Lemma 25.5. *There exists universal constants $d \in \mathbb{N}$, $\gamma > 0$, and $\beta_1 > 0$ for which the following holds. Given an instance of graph CSP G, one can compute an instance of graph CSP G' with the following properties.*

1. *The graph supporting G' is a d-regular undirected graph.*
2. *G' has the same alphabet size as G.*
3. *$\beta_1 \,\textsc{unsat}(G) \leq \textsc{unsat}(G') \leq \textsc{unsat}(G)$.*
4. *The spectral gap on G' is $\geq \gamma$.*
5. *The size of G' is at most a constant factor greater than the size of G.*

We prove Lemma 25.5 in Section 25.4.

Error Amplification. The next step, powering, is where we amplify UNSAT. Recall that in normal graphs, taking the kth power means we generate an edge for every k-edge walk in the input graph. The resulting graph is denser and the spectral gap is larger.

Powering a graph CSP is similar in spirit but more complicated since we must address the CSP aspects as well. Here we give a high level sketch. The underlying graph will be the kth graph power. Each edge $\{u, v\}$ in the powered graph corresponds to a k-edge walk in the original graph, which corresponds to k binary constraints. Loosely speaking, these k constraints are combined into one large constraint that in some sense requires all k constraints to be satisfied simultaneously. Intuitively, this increases the unsatisfiability since we now have to simultaneously satisfy k clauses in the input graph CSP to satisfy just 1 clause in the powered graph CSP.

For this idea to make syntactic sense, we have to increase the alphabet so that for each "power constraint" of k input clauses, each vertex has "k letters" to be supplied to each of the clauses. Thus the alphabet expands from A to A^{d^k}; i.e., d^k letters per vertex. These d^k labels for a vertex v are interpretted as labeling the entire k-step neighborhood of v.

Again, this is only a high-level description, and we explain the construction much more carefully in Section 25.5. All put together, the bundled constraints ramp up UNSAT, but we pay the price of a much larger alphabet size.

Lemma 25.6. *Let $d, \gamma, |A|$ be fixed. Then there exists $\beta_2 > 0$ for which the following holds for all $t = 2s + 1$ where $s \in \mathbb{N}$. Let G be a regular graph CSP over an alphabet A with degree d and spectral gap (at least) γ. Then one can compute a graph-CSP G^t with the following properties.*

1. *G^t is regular with degree d^t, alphabet A^{d^s}, and the random walk on G^t has spectral gap $1-(1-\gamma)^t$.*
2. *If $\textsc{unsat}(G)=0$, then $\textsc{unsat}(G^t)=0$.*
3. *$\textsc{unsat}(G^t) \geq \beta_2\sqrt{t}\min\left\{\textsc{unsat}(G), \frac{1}{t}\right\}$.*

Like many of our recent discussions, this will be based on analyzing random walks on G, and here we will see a dependence on the spectral gap γ in the parameter β_2.

Alphabet reduction. The third step addresses the blow up in alphabet size from powering the graph CSP.

Lemma 25.7. *There exists constants $C \in \mathbb{N}$ and $\beta_3 \in (0,1)$ for which the following holds. Given a regular graph CSP G with alphabet A, one can compute a graph CSP with size $f(|A|)|G|$ such that*

$$\beta_3\,\textsc{unsat}(G) \leq \textsc{unsat}(G') \leq \textsc{unsat}(G).$$

Putting it all together. All put together, these three operations take as input one graph CSP G over a constant-sized alphabet A_0 and outputs another graph CSP G' over the same alphabet. For a parameter $t \in \mathbb{N}$, if $\textsc{unsat}(G)$ is a universal constant factor smaller than t, then $\textsc{unsat}(G')$ is at most a universal constant factor

Step	input		expander-ification		powering		alphabet reduction
Lemma			25.5		25.6		25.7
Degree	arbitrary	$\rightarrow$	d_0	$\rightarrow$	d_0^t	$\rightarrow$	arbitrary
γ	arbitrary	$\rightarrow$	γ_0	$\rightarrow$	$1-(1-\gamma_0)^t$	$\rightarrow$	arbitrary
$\textsc{unsat}(G)$	α	$\rightarrow$	$\beta_1\alpha$	$\rightarrow$	$\sqrt{t}\beta_1\beta_2\alpha,$	$\rightarrow$	$\sqrt{t}\beta_1\beta_2\beta_3\alpha$
Alphabet	A_0	$\rightarrow$	A_0	$\rightarrow$	$A_0^{d_0^t}$	$\rightarrow$	A_0

Figure 25.1: In the row for $\textsc{unsat}(G)$, we assume that $\beta_1\alpha \leq \frac{1}{t}$, since otherwise $\textsc{unsat}(G)$ is at least a constant and we are done.

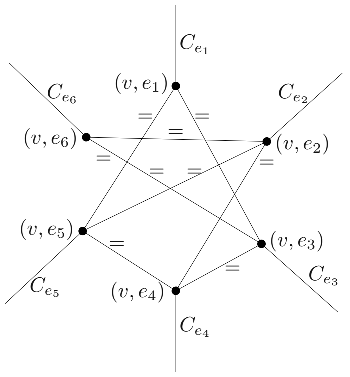
smaller than $\sqrt{t}\,\textsc{unsat}(G)$. (See Fig. 25.1.) By making t a sufficiently small constant, this means that if $\textsc{unsat}(G)$ is smaller than a universal constant, than $\textsc{unsat}(G')$ is greater than $\textsc{unsat}(G)$ by a universal constant > 1. That is, we've successfully amplified UNSAT without increasing the input size.

25.4 Expander-ification

The first transformation preprocesses the graph-CSP to be a constant degree expander. We proceed in two stages where the first stage addresses the degree and the second stage addresses the spectral gap.

To make the degree small and uniform, we replace each vertex $v \in V$ with a low degree expander as follows.

1. For each vertex v and each edge e incident to v, we create a new vertex (v, e).
2. For each edge $e = (u, v)$, we have an edge from (u, e) to (v, e) with the same constraint as e.
3. Fix v. We currently have an auxiliary vertex (v, e) for every edge e incident to v. Let G_0 be a d_0-regular expander with vertex set $\{(v, e)\}$. For each edge e in the expander, we assign the "equality constraint" $C_e = \{(a, a) : a \in A\} \subset A^2$.



Exercise 25.3. Let G' be the graph CSP obtained from G via steps 1–3 above. Prove that G' has the following properties:

(a) The total number of edges of G' is within a constant factor of the number of edges of G.
(b) G' has the same alphabet as G.
(c) G' is a d-regular graph for a universal constant d.
(d) $c\,\textsc{unsat}(G) \leq \textsc{unsat}(G') \leq \textsc{unsat}(G)$ for some universal constant c.

We now assume that $G = (V, E)$ is regular with constant degree $d = O(1)$. We now make G an expander while retaining the degree. This is done by simply overlaying another expander

Let $H = (V, E_H)$ be a constant degree expander on the same vertex set V. For all $e \in E_H$, let C_e be the trivial constraint that allows for all pairs of labels. Let $G' = (V, E + E_H)$ be the graph-CSP combining the clauses from G and H. The G' is an expander while retaining the salient properties of G, which is left to the reader to show.

Exercise 25.4. Prove the following properties about the graph-CSP $G' = (V, E + E_H)$ described on the previous page.

(a) G' has constant degree.
(b) G' has constant spectral gap γ.
(c) The total size of G' is at most a constant factor greater than G'.
(d) $\Omega(\textsc{unsat}(G)) \leq \textsc{unsat}(G') \leq \textsc{unsat}(G)$.

Combining our two preprocessing steps gives Lemma 25.5.

25.5 Error amplification

We now turn to the error amplification stage. The input is a graph-CSP G where the underlying graph is a regular expander with constant degree d. The goal is to produce a new graph-CSP G' that (a) is satisfiable iff G is satisfiable, and (b) has substantially larger $\textsc{unsat}(G')$ when G is not satisfiable. Thanks to the preceding preprocessing step, we can assume that G is a d-regular expander. We also assume that $|A|$ is a fixed constant.

Dinur [Din07] offers the following helpful intuition. Fix an assignment $\pi : V \to A$. Suppose we sample t edges in G uniformly at random and test if any of the corresponding constraints are not satisfied. The probability that at least one of them is unsatified is $1 - (1 - \textsc{unsat}(G))^t \approx t\,\textsc{unsat}(G)$ (for $\textsc{unsat}(G)$ small). To embed this logic into a CSP, consider the (non-binary) CSP where for every t edges $e_1, \ldots, e_t$ of G, we make a constraint over the (at most) $2t$ underlying vertices, which is satisfied iff all t constraints at $e_1, \ldots, e_t$ are satisfied. This CSP will have $\textsc{unsat}$ value $1 - (1 - \textsc{unsat}(G))^t \approx t\,\textsc{unsat}(G)$, which is good. However, (a) the resulting CSP is no longer binary, and (b) the number of constraints in the CSP increases substantially from m to roughly m^t.

In short, we can increase $\textsc{unsat}$ by independent repetition, but encoding this directly as a graph-CSP is inefficient. We want to replicate the overall effect but in a more efficient and graph-friendly manner. Recall that an expander graph mixes rapidly, and a random walk on an expander graph behaves similarly to uniform sampling. Here (the underlying graph of) G is a constant degree expander. Since G is an expander, then the t-step walks should behave like independent samples of t edges. We will create a graph-CSP instance on top of the t-th graph power of G.

Powering a graph-CSP. We sketched the construction in Section 25.3 and now we describe it in full detail. Consider the graph G^t that has edges corresponding to *lazy* t-step random walks. More precisely, let us generate for each vertex v, d self-loops at G. Call this graph G_L; a random t-step walk in G_L corresponds to a lazy t-step walk in G. We create an edge (v_0, v_t) in G^t for every t-step walk $(v_0, \ldots, v_t)$ in G_L.

For a vertex v, let $N^t(v)$ denote the set of vertices within t steps of v in G (including v). We have

$$\left|N^t(v)\right| \leq D \text{ where } D = \sum_{i=0}^{t} d^i = (1+d)^t.$$

We use the alphabet A^D. We identify an A^D-label $\bar{\pi}(v) \in A^D$ as an A-labeling of all of $N^t(v)$. We let $\bar{\pi}(v,w) \in A$ denote the label assigned by $\bar{\pi}(v)$ to $w \in N^t(v)$.

For each t-step walk $w = (v_0, \dots, v_t)$ in G_L, which corresponds to a distinct edge e_w between v_0 and v_t in G^t, we create a constraint C_w that is satisfied by $\bar{\pi}(v_0), \bar{\pi}(v_t) \in A^D$ iff the following holds.

(a) For each edge $e = (v_i, v_{i+1}) \in w$, the labels $(\bar{\pi}(v_0, v_i), \bar{\pi}(v_t, v_{i+1})) \in A^2$ satisfies the constraint C_e in G.

(b) For all i, $\bar{\pi}(v_0, v_i) = \bar{\pi}(v_t, v_i)$.

Property (a) is where we must satisfy all t underlying constraints. Property (b) forces v_0 and v_t to agree on all the vertices along the walk.

Analysis. It is easy to see that a satisfying assignment for G gives a satisfying assignment for G^t. In the converse direction, we want to show that if G has nonzero $\text{UNSAT}(G)$, then G^t has substantially larger $\text{UNSAT}(G)$. In particular we will show that $\text{UNSAT}(G^t) \geq \text{poly}(t)\,\text{UNSAT}(G)$.

We prove this via the contrapositive: given any labeling $\bar{\pi} : V \to A^D$, we derive a labeling $\pi : V \to A$ and show that $\text{UNSAT}(\pi \mid G) \leq \text{poly}(1/t)\,\text{UNSAT}(\bar{\pi} \mid G^t)$. The labeling π is derived from $\bar{\pi}$ as follows.

Fix $v \in V$. Let $v_0 = v, v_1, \dots$ denote a lazy random walk starting from v. Consider the random label $\bar{\pi}\left(v_{t/2}, v\right)$. We define $\pi(v)$ to be the most likely label to arise as $\bar{\pi}\left(v_{t/2}, v\right)$. We have

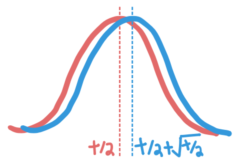
$$\mathbf{P}\left[\bar{\pi}\left(v_{t/2}, v\right) = \pi(v)\right] \geq 1/|A|, \tag{25.1}$$

which as far as we're concerned, is at least a constant. Up to constants, (25.1) extends to indices close to $t/2$. Let $I = \left\{i : |i - t/2| \leq \sqrt{t/2}\right\}$.

Lemma 25.8. *For sufficiently large t, there exists a universal constant $c > 0$ such that for all $i \in I$,*

$$\mathbf{P}[\bar{\pi}(v_i, v) = \pi(v)] \geq c/|A|.$$

We briefly sketch the intuition and postpone the formal proof to the end of the section. Because all $i \in I$ are very close to $t/2$, the number of "non-lazy" steps j in an i-step random walk is distributed almost the same for all $i \in I$. Conditional on j, $\bar{\pi}(v_i, v)$ is independent of i. Putting these two facts together means that $\bar{\pi}(v_i, v)$ is distributed very similarly for all i, and in particular, similar to $\bar{\pi}\left(v_{t/2}, v\right)$.



Before proceeding to the main part of the proof, we require one more technical lemma that brings into play the assumption that G is an expander. Recall that the edges along a random walk in an expander behave almost independently. This is reflected in the following lemma.

Lemma 25.9. *Let $F \subset E$ be any subset of edges, and let $\mu = |F|/|E|$. Consider a random walk $v_0, v_1, v_2, \ldots$ in G, where initially v_0 is a uniformly random endpoint of a uniformly random edge from F. Then*

$$\mathbf{P}[(v_i, v_{i+1}) \in F] \leq \mu + (1-\gamma)^i.$$

for all i, where γ is the spectral gap of G.

We postpone the proof of this lemma as well until the end of the section, and instead turn to the main claim. In the following, note that we will take t to be a constant, so if $\textsc{unsat}(\pi \mid G) \geq 1/t$ then we have already accomplished our main goal.

Lemma 25.10. *$\textsc{unsat}(\bar{\pi} \mid G^t) \geq \Omega\left(\sqrt{t}\right) \min\left\{\textsc{unsat}(\pi \mid G), \frac{1}{t}\right\}$.*

Proof. Let $F \subset E$ be the set of edges failed by π, and let $\mu = |F|/|E| = \textsc{unsat}(\pi \mid G)$. If $\mu \geq 1/t$, then drop edges from F until $\mu \leq 1/t$. We have $1/t \geq \mu \geq \Omega(1) \min\left\{\textsc{unsat}(\pi \mid G), \frac{1}{t}\right\}$.

Let $w = (v_0, \ldots, v_t)$ be a lazy t-step walk in G sampled uniformly at random. For each $i \in I$, we define a random indicator variable $X_i \in \{0,1\}$ where $X_i = 1$ if, letting $e_i = (v_{i-1}, v_i)$ be the ith edge in w,

(a) The ith edge in w, $e_i = (v_{i-1}, v_i)$, has labels $\bar{\pi}(v_{i-1}, v_0) = \pi(v_{i-1})$ and $\bar{\pi}(v_t, v_i) = \pi(v_i)$.

(b) Either e_i is a loop or π fails the constraint C_{e_i}.

Let $X = \sum_{i \in I} X_i$. Observe that $\textsc{unsat}(G^t) \geq \mathbf{P}[X > 0]$. We make two claims about X.

Claim 1. For all $i \in I$, $\mathbf{E}[X_i] \geq \Omega(1)\mu$.

Claim 2. $\mathbf{E}\left[X^2\right] \leq O\left(\sqrt{t}\right)\mu$.

We will prove the claims later. Let us first assume they hold and complete the proof. Linearity of expectation and claim 1 gives $\mathbf{E}[X] \geq \Omega\left(\sqrt{t}\right)\mu$. We also have

$$\mathbf{E}[X]^2 = \mathbf{E}[X \mid X > 0]^2\,\mathbf{P}[X > 0]^2 \overset{(a)}{\leq} \mathbf{E}\left[X^2 \,\middle|\, X > 0\right]\mathbf{P}[X > 0]^2 = \mathbf{E}\left[X^2\right]\mathbf{P}[X > 0]$$

by (a) Jensen's inequality (for the convex function $f(x) = x^2$.) Rearranging and applying the claims, we have

$$\mathbf{P}[X > 0] \geq \frac{\mathbf{E}[X]^2}{\mathbf{E}[X^2]} \geq \Omega\left(\sqrt{t}\right)\mu,$$

as desired.

It remains to prove the claims, starting with claim 1. Suppose we sample X_i alternatively as follows. Sample an edge (v_{i-1}, v_i) uniformly at random. Take lazy random walks $v_{i-1}, \ldots, v_0$ and $v_i, \ldots, v_t$, and output the walk $w = (v_0, \ldots, v_t)$. This produces a uniformly random lazy walk because G is regular. We have

$$\mathbf{P}[X_i = 1] = \mu \cdot \mathbf{P}[\bar{\pi}(v_0, v_{i-1}) = \pi(v_{i-1})]\,\mathbf{P}[\bar{\pi}(v_t, v_i) = \pi(v_i)].$$

Recall from Lemma 25.8 that, for $i \in I$, the marginal probabilities of $\bar{\pi}(v_t, v_i)$ are within a constant factor of the marginal probabilities of $\pi\left(v_t, v_{t/2}\right)$, which in turn is at least $1/|A|$. Similarly for $\bar{\pi}(v_0, v_i)$. Thus

$$\mathbf{P}[X_i = 1] \geq \Omega(1)\mu/|A|^2 = \Omega(1)\mu,$$

as desired.

The remaining claim, claim 2, is about the variance of $X = \sum_{i \in I} X_i$. To this end we have an intermediate claim analyzing the cross-terms $X_i X_j$.

Claim 3. Let $i, j \in I$ with $i \neq j$. Then $\mathbf{E}[X_i X_j] \leq \mu\left(\mu + (1-\gamma)^{j-i-1}\right)$.

Let us define random indicators variables $Y_i, Y_j \in \{0, 1\}$ that indicate whether the ith edge is in F. Then $0 \leq X_i \leq Y_i$ hence $\mathbf{E}[X_i X_j] \leq \mathbf{E}[Y_i Y_j]$. Write

$$\mathbf{E}[Y_i Y_j] = \mathbf{E}[Y_i]\,\mathbf{E}[Y_j \mid Y_i = 1] = \mu\,\mathbf{E}[Y_j \mid Y_i].$$

The remaining term, $\mathbf{E}[Y_j \mid Y_i = 1]$, is equivalent to the probability that a random walk starting at a random endpoint of a uniformly random edge in F takes its $(j-i)$th step in F. By Lemma 25.9, this probability is at most $\mu + (1-\gamma)^{j-i-1}$.

Now we prove claim 2. We have

$$\begin{aligned}\mathbf{E}\left[X^2\right] &= \sum_{i \in I} \mathbf{E}[X_i] + 2 \sum_{\substack{i,j \in I \\ i<j}} \mathbf{E}[X_i X_j] \\ &\leq \mu|I| + 2\mu^2|I|^2 + 2\mu \sum_{\substack{i,j \in I \\ i<j}} (1-\gamma)^{j-i-1} \\ &\overset{\text{(b)}}{=} \Omega(1)\mu|I| + 2\mu^2|I|^2 \overset{\text{(c)}}{\leq} \Omega\left(\sqrt{t}\right)\mu.\end{aligned}$$

Here (b) is by claim 3. (c) is because $\mu \geq 1/t$ and $|I| = O\left(\sqrt{t}\right)$.

This completes the proof of Lemma 25.10. ☐

This establishes, modulo Lemma 25.8 and Lemma 25.9 which were introduced earlier in the section. The remainder of this section is devoted to proving these lemmas.

An expander mixing lemma for edges. Let us first prove Lemma 25.9 since it is arguably more interesting. In particular, it reveals why it is important that G is an expander graph. We briefly recall the motivation. We want to argue that the failed edges F are not too correlated along random walks. This is because if they are correlated, then the number of bad edges per walk, X in our high analysis, will have high variance (and in particular, claim 2 will fail).

Lemma 25.9. *Let $F \subset E$ be any subset of edges, and let $\mu = |F|/|E|$. Consider a random walk $v_0, v_1, v_2, \ldots$ in G, where initially v_0 is a uniformly random endpoint of a uniformly random edge from F. Then*

$$\mathbf{P}[(v_i, v_{i+1}) \in F] \leq \mu + (1-\gamma)^i.$$

for all i, where γ is the spectral gap of G.

Proof. Let G be a regular undirected graph and let R be the random walk map. Let $x \in \Delta^V$ be the initial distribution for v_0. For each vertex v, we have

$$x(v) = \frac{(\#\text{ edges in } F \text{ incident to } v)}{2|F|}.$$

Note that $x(v) \leq (d/2|F|)$ for all v. Define $y : V \to [0,1]$ be setting, for each $v \in V$,

$$y(v) = \frac{(\#\text{ edges in } F \text{ incident to } v)}{d} = \left(\frac{2|F|}{d}\right) x(v).$$

This is equal to the probability that a random step from v is in F. The probability that the tth step is in F is exactly

$$\mathbf{P}[(v_{t-1}, v_t) \in F] = \left\langle y, R^{t-1} x \right\rangle = \frac{2|F|}{d} \left\langle x, R^{t-1} x \right\rangle.$$

Since G is a regular undirected graph with spectral gap γ, we can write

$$R^{t-1} = \frac{1}{n}(\mathbb{1} \otimes \mathbb{1}) + R'$$

where $\|R'\| \leq \gamma^{t-1}$. The remainder of the proof is left to the reader in the following exercise. $\square$

Exercise 25.5. Complete the proof above via the following steps.

1. Prove that $\langle x, R^{t-1} x\rangle \leq \frac{1}{n} + \frac{\gamma^{t-1} d}{2|F|}$.
2. Prove that $\mathbf{P}[(v_{t-1}, v_t) \in F] \leq \frac{|F|}{|E|} + \gamma^{t-1}$.

Similarity of lazy random walks. The final lemma to prove is about how lazy i-step walks act similarly for different $i \in I$. We start by stating the following lemma about binomial distributions.

Lemma 25.11. *For every $c > 0$, there exists some constant $z \in (0,1)$ and n_0 such that, if $n_0 < n - \sqrt{n} \leq m < n + \sqrt{n}$, then for all k such that $|k - n/2| \leq c\sqrt{m}$, we have*

$$z \leq \frac{\mathbf{P}[B_n = k]}{\mathbf{P}[B_m = k]} \leq \frac{1}{z}$$

The proof of Lemma 25.11 is given as exercise 25.7. Let us continue to prove Lemma 25.8.

Lemma 25.8. *For sufficiently large t, there exists a universal constant $c > 0$ such that for all $i \in I$,*

$$\mathbf{P}[\bar{\pi}(v_i, v) = \pi(v)] \geq c/|A|.$$

Proof. For $i \in \mathbb{N}$, let B_i be the number of non-lazy steps out of the first k steps in the lazy random walk. Choose $c > 0$ such that the probability that $\left|B_{t/2} - t/4\right| \geq c\sqrt{t} \leq 1/2|A|$. For ease of notation, let E_i be the event that $\pi(v_i \mid v_0) = \pi(v_0)$. We want to

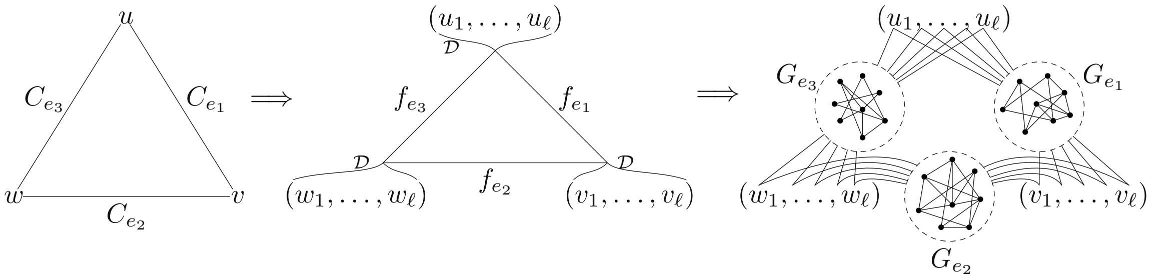


Figure 25.2: A high-level schematic for the alphabet reducing transformation in Section 25.6, on a triangle graph.

show that for $i \in I$, we have $\mathbf{P}[E_i] \geq \Omega(1)\,\mathbf{P}\left[E_{t/2}\right]$. Let $J = \left\{ j : |j - t/4| \leq c\sqrt{t/2} \right\}$ We have

$$\begin{aligned}
\mathbf{P}[E_i] &\geq \sum_{j \in J} \mathbf{P}[E_i \mid B_i = j]\,\mathbf{P}[B_i = j] \\
&\overset{(a)}{\geq} z \sum_{j \in J} \mathbf{P}[E_i \mid B_i = j]\,\mathbf{P}\left[B_{t/2} = j\right] \\
&\overset{(b)}{=} z \sum_{j \in J} \mathbf{P}\left[E_{t/2} \,\middle|\, B_{t/2} = j\right]\mathbf{P}\left[B_{t/2} = j\right] \\
&\overset{(c)}{\geq} z\left(\mathbf{P}\left[E_{t/2}\right] - \frac{1}{2|A|}\right) \overset{(d)}{\geq} \frac{z}{2}\,\mathbf{P}\left[E_{t/2}\right],
\end{aligned}$$

as desired. Here (a) is by Lemma 25.11, for t sufficiently large and z the constant asserted in Lemma 25.11. The reason for (b) is left as an exercise below. (c) is by choice of c. (d) is because $\mathbf{P}\left[E_{t/2}\right] \geq 1/|A|$. □

Exercise 25.6. Justify equality (b) in the proof of Lemma 25.8.

25.6 Alphabet reduction

This section is about the third graph-CSP transformation reduction, where the goal is to reduce the size of the alphabet. We take as input a graph CSP G with alphabet A. Our goal is to reduce the alphabet to an alphabet A_0 where $|A_0|$ is a universal constant. (In fact, the full details reveal that $|A_0| = 8$.)

Step 1: Reduce the alphabet size to 2 with an error correcting code. Let $\mathcal{C} : A \to \{0,1\}^\ell$ be an error correcting code with $\ell \leq O(\log(|A|))$ and relative distance $\rho \in [0,1]$; that is, for any distinct $a_1, a_2 \in A$, the encodings $\mathcal{C}(a_1), \mathcal{C}(a_2) \in \{0,1\}^\ell$

differ in at least $\rho\ell$ bits. We will create a uniform (2ℓ)-ary CSP with binary alphabet $\{0,1\}$. Such a CSP can be interpreted as a (2ℓ)-uniform hypergraph with hyperedges labeled by boolean formulas over the endpoints defining a constraint.

For each vertex u, let $V_u = \{u_1, \dots, u_\ell\}$ be a new set of ℓ vertices. For each edge $e = (u,v) \in E$ with constraint C_e, we create a hyperedge with endpoints $V_u \cup V_v$ with the constraint $f_e : \{0,1\}^{V_u} \times \{0,1\}^{V_v} \to \{0,1\}$ defined by

$$f_e(x_u, x_v) = \begin{cases} 1 & \text{if } x_u = \mathcal{C}(a_1) \text{ and } x_v = \mathcal{C}(a_2) \text{ for some } (a_1,a_2) \in C_e, \\ 0 & \text{otherwise;} \end{cases}$$

where we identify a boolean assignment $x_u : V_u \to \{0,1\}$ as a length ℓ bit string $(x_u(u_1), \dots, x_u(u_\ell))$. We let $F_e = \{(x_u,x_v) : f_e(x_u,x_v) = 1\} \subset \{0,1\}^{V_u \times V_v}$ denote the set of satisfying assignments for f_e.

Step 2: Replace each hyperedge constraint with a graph CSP. The next step converts the hypergraph generated above into a graph. We will apply a certain operation hyperedge-wise that reduces each hyperedge independently to a graph CSP. Analyzing this operation requires new (Fourier-analytic) techniques that are better to discuss separately (Chapter 27). For now we will just use it as a black box.

Let $\epsilon_0 > 0$ be a parameter A_0 be a finite alphabet of cardinality to be determined. For each formula f_e, we construct a graph-CSP $G_e = (V_e, E_e, A_0, \{C_{e,f} : f \in E_e\})$, where $V_u, V_v \subseteq V_e$, and which has the following properties.

1. *Completeness*: If $f_e(x_u,x_v) = 1$, then we can extend $(x_u,x_v) : V_u \cup V_v \to \{0,1\}$ to an assignment $\sigma : V_e \to A_0$ that satisfies all of G_e.
2. *Soundness*: If $f_e(x_u,x_v) = 0$, then any extension $\sigma : V_e \to A_0$ of (x_u,x_v) has error proportional to the relative Hamming distance between (x_u,x_v) and F_e.

$$\textsc{unsat}(\sigma \mid G_e) \geq \epsilon_0 \operatorname{dist}((x_u,x_v), F_e)$$

for all extensions σ of (x_u,x_v), where $\operatorname{dist}(a,b)$ denotes the relative Hamming distance between a and b.

Moreover, the parameter ϵ_0 and the size of A_0 depending only on $|\ell|$. The size of $|E_e|$ is the same for all e.

In lieu of a proper analysis, we provide some high-level comments. For each edge e, we transform a boolean function on 2ℓ variables to a graph CSP with finite alphabet size and possibly a few more variables. There is no particular concern for the size of the graph CSP as long as it is consistent across E. Why? Because it is a *local* operation applied edgewise. The input f_e ultimately can be described in $\tilde{O}\left(2^\ell\right)$ bits, *and ℓ is independent of n.* So however much things may blow up locally, we are still blowing up a constant to a constant.

The above construction, that is applied to each boolean function, can be understood as an *inefficient* and *distance-preserving* PCP.

Analysis. Thus we have a two-step process that takes one graphical CSP and produces another with a constant alphabet size. Between the two steps we have a hypergraph CSP over the alphabet $\{0,1\}$. We assume that step 2 is implementable, and defer those details to Chapter 27.

For each e, let $V'_e = V \setminus (V_u \cup V_v)$ be the set of variables introduced by G_e. Let $V' = \bigcup_e V'_e$ be the set of all newly introduced variables.

Lemma 25.12. *$\textsc{unsat}(\bar{G}) \leq \textsc{unsat}(G)$.*

Proof. Let $\pi : V \to A$ be an assignment that attains $\textsc{unsat}(G)$. Recall that the vertices of G' can be divided into vertices V_u (where $u \in V$) corresponding to the input vertices, and vertices V'_e (where $e \in E$) introduced by the edge-wise CSP's $G_e : e \in E$.

1. For each $u \in V$, we label V_u with the encoding $\mathcal{C}(\pi(u))$.
2. For each V'_e where $e = (u,v) \in E$, given the labels already on V_u and V_v, label V'_e as to maximize the number of satisfied constraints in G_e.

Now, for every edge e satisfied by π, all the constraints from G_e are satisfied by the labels in $\bar{V}$. Since each G_e generates the same number of constraints, this implies that $\textsc{unsat}(\bar{G}) \leq \textsc{unsat}(G)$. □

Lemma 25.13. *For $\beta_3 = \epsilon_0 \rho / 4$, we have*

$$\beta_3 \, \textsc{unsat}(G) \leq \textsc{unsat}(\bar{G}).$$

Proof. Let $\bar{\pi} : \bar{V} \to A_0$ be an assignment for $\bar{\pi}$ that attains $\textsc{unsat}(\bar{G})$. Let $\pi : V \to A$ be the assignment defined by decoding; for each vertex v, we have

$$\pi(v) = \mathcal{D}(\bar{\pi}(v_1), \ldots, \bar{\pi}(v_\ell)).$$

We claim that *for each edge e that is not satisfied by π, a β_3-fraction of G_e is unsatisfied for some value β_3 that depends on $|A_0|$ and ρ.* It then follows that $\textsc{unsat}(\bar{G}) \geq \beta_3 \, \textsc{unsat}(G)$.

Let $e = (u, v)$ be an edge that is not satisfied by π. For ease of notation, let us denote

$$\bar{u} \overset{\text{def}}{=} (\bar{u}_1, \dots, \bar{u}_\ell) \text{ and } \bar{\pi}(\bar{u}) \overset{\text{def}}{=} (\bar{\pi}(u_1), \dots, \bar{\pi}(u_\ell)).$$

Since $(\pi(u) = \mathcal{D}(\bar{u}), \pi(v) = \mathcal{D}(\bar{v}))$ did not satisfy the constraint C_e, and the code $\mathcal{C} : A \to \{0,1\}^\ell$ has relative distance ρ between any two code works, it follows that $(\bar{\pi}(\bar{u}), \bar{\pi}(\bar{v}))$ relative distance at least $\rho/4$ from the set of satisfying encodings, F_e. By the soundness property, then, $\bar{\pi}$ must fail to satisfy at least an $(\epsilon_0\rho/4)$-fraction of G_e. □

25.7 Additional notes and materials

See [AB09, Chapter 11] for further background on the PCP theorem.

Spring 2024 lecture notes. Click on the links below for the following files:

- Handwritten notes prepared before the lecture. (Part 1)
- Handwritten notes annotated during the presentation. (Part 1)
- Handwritten notes prepared before the lecture. (Part 2)
- Handwritten notes annotated during the presentation. (Part 2)
- Recorded video lecture.

Fall 2022 lecture materials, part 1. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

Fall 2022 lecture materials, part 2. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

25.8 Exercises

Exercise 25.1. Prove that $\mathbf{PCP}(O(\log n), O(1)) \subseteq \mathbf{NP}$.

Exercise 25.2. Prove Theorem 25.1 and Theorem 25.3 are equivalent. Below we give part of the proofs, in both directions, to get you started.

1. *Theorem 25.1* $\implies$ *Theorem 25.3.* Suppose the PCP theorem, Theorem 25.1, is true. That is, every NP language L has a verifier on input x and proof y that reads $r = c\log n$ random bits and querys $q = O(1)$ bits from y, and correctly. We want to show that (1/2)-approximate for CSP — that is, deciding between whether a CSP is (perfectly) satisfiable or if at most 1/2 of the clauses can be satisfied — is NP-Hard.

   Fix a language L in NP. Given input x of size n, we want to form a CSP problem P such that deciding between $\text{UNSAT}(P) = 0$ and $\text{UNSAT}(P) \geq 1/2$ is equivalent to deciding if $x \in L$. By the PCP theorem, there exists a verifier that flips at most $r = c\log(n)$ coins and reads $q = O(1)$ bits from the proof and decides whether to accept or reject. Let $A = \{0, 1\}$ be the alphabet, and make a boolean variable v_i for every location i of the proof that might be accessed by the randomized verifier. Note that this creates at most $q2^r = \text{poly}(n)$ boolean variables. Now, for each $z \in \{0,1\}^r$, representing an outcome of the coin tosses, we defined a clause C_z with variables... and accepting the set of assignments...

2. *Theorem 25.3* $\implies$ *Theorem 25.1.* Conversely, suppose that it is NP-Hard to decide between $\text{UNSAT}(P) = 0$ and $\text{UNSAT}(P) \geq 1/2$ for a given CSP problem P. This means that for every language L, there is a transformation that, given an input x of size n, produces a q-ary CSP P_x with poly(n) constraints such that $x \in L$ iff $\text{UNSAT}(P_x) = 0$ and $x \notin L$ iff $\text{UNSAT}(P_x) \geq 1/2$. We create a probabilisticaly checkable proof system where...

Exercise 25.3. Let G' be the graph CSP obtained from G via steps 1–3 on page 328. Prove that G' has the following properties:

(a) The total number of edges of G' is within a constant factor of the number of edges of G.
(b) G' has the same alphabet as G.
(c) G' is a d-regular graph for a universal constant d.
(d) $c\,\text{UNSAT}(G) \leq \text{UNSAT}(G') \leq \text{UNSAT}(G)$ for some universal constant c.

Exercise 25.4. Prove the following properties about the graph-CSP $G' = (V, E + E_H)$ described on page 328.

(a) G' has constant degree.
(b) G' has constant spectral gap γ.
(c) The total size of G' is at most a constant factor greater than G'.
(d) $\Omega(\text{UNSAT}(G)) \leq \text{UNSAT}(G') \leq \text{UNSAT}(G)$.

Exercise 25.5. Complete the proof above via the following steps.

1. Prove that $\langle x, R^{t-1}x\rangle \leq \frac{1}{n} + \frac{\gamma^{t-1}d}{2|F|}$.

2. Prove that $\mathbf{P}[(v_{t-1}, v_t) \in F] \leq \frac{|F|}{|E|} + \gamma^{t-1}$.

Exercise 25.6. Justify equality (b) in the proof of Lemma 25.8.

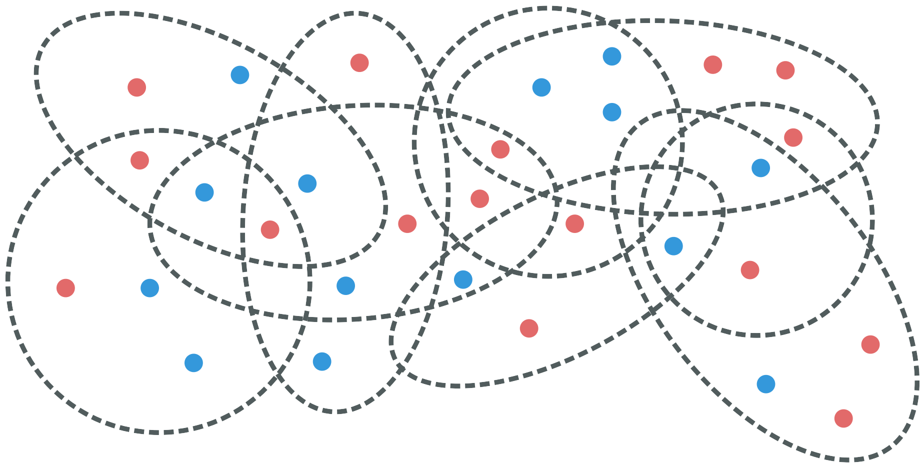
Exercise 25.7. For $n \in \mathbb{N}$, let B_n be a binomially distributed random variable with probability $p = 1/2$. Prove the following.

For every $c > 0$, there exists some constant $z \in (0,1)$ and n_0 such that, if $n_0 < n - \sqrt{n} \leq m < n + \sqrt{n}$, then for all k such that $|k - n/2| \leq c\sqrt{m}$, we have

$$z \leq \frac{\mathbf{P}[B_n = k]}{\mathbf{P}[B_m = k]} \leq \frac{1}{z}$$

Chapter 26

Discrepancy via Gaussian random walks



26.1 Balanced coloring

Let $S_1, \ldots, S_m \subseteq [n]$ be a collection of m sets of a universe of n points. We would like to color all the points in one of two colors so that each set has the same number of points of each color, or as close to the same as possible.

More formally, we encode colorings as vectors $x \in \{-1, 1\}^n$ where -1 indicates one color and $+1$ indicates the other. For a set S_i, let

$$x(S_i) \overset{\text{def}}{=} \sum_{j \in S_i} x_j$$

denote the sum over the coordinates in S_i. The *discrepancy* of S_i is defined as absolute difference between the number of points of each color,

$$|x(S_i)| = \left|\sum_{j \in S_i} x_j\right|.$$

The goal is to color the points as to minimize the maximum discrepancy:

$$\min_{x \in \{-1,1\}^n} \max_{S_i} |x(S_i)|.$$

The additive Chernoff bound and union bound imply that a uniformly random $z \in \{-1,1\}^n$ has maximum discrepancy at most $O\left(\sqrt{n \log(m)}\right)$. (Exercise 26.1.)

Surprisingly, for the special case $m = n$, Spencer [Spe85] showed that there exists a coloring $z \in \{-1,1\}^n$ with discrepancy

$$O\left(\sqrt{n}\right),$$

erasing the logarithmic factor. For $m > n$ the bound becomes

$$O\left(\sqrt{n \log(m/n)}\right)$$

However the proof was existential in nature and it remained an open to find such a coloring efficiently.

In 2010, Bansal [Ban10] discovered an algorithm that achieves discrepancy $O(\sqrt{n} \log(m/n))$; in particular this gives $O(\sqrt{n})$ for the symmetric setting $m = n$. Subsequent work by Lovett and Meka [LM12] gave an elegant randomized polynomial time algorithm achieving the more general bound of $O\left(\sqrt{n \log(m/n)}\right)$. This second algorithm is the main topic of this lecture. Formally, we will prove the following.

Theorem 26.1. *Given m sets $S_1, \ldots, S_m$ over n points, there exists a vector $z \in \{-1,1\}^n$ such that $z(S_i) \leq O\left(\sqrt{n \log(m/n)}\right)$ for all i. This vector z can be computed in randomized polynomial time in expectation.*

26.2 Reduction to partial coloring

A *fractional coloring* is defined as a vector $x \in [-1,1]^n$. The algorithm maintains a fractional coloring $x \in [-1,1]^n$, initially set to $x = \mathbb{0}$, that meets the discrepancy bound $|x(S_i)| \leq O\left(\sqrt{n \log(m/n)}\right)$ for all i, at all times. Progress is made by having the coordinates x_j approach the integral extremes $\{-1,1\}$ over time.

Let $\epsilon = 1/n$. We say that a coordinate x_j is *ϵ-tight*, or simply *tight*, if $|x_j| \geq 1 - \epsilon$. Otherwise we say that x_j is *free*. We say that x is tight if all the coordinates are tight. If x is tight, then it is easy to see that we can deterministically round x to a proper coloring in $\{-1, 1\}^n$ without significantly increasing the discrepancy. Thus the task is reduced to computing a tight fractional coloring x.

We break down this task further into *partial coloring* problems, where the goal is to obtain a fractional coloring x with good discrepancy and where *half* the coordinates are tight. The tight coordinates are then fixed forever, and in the next iteration we apply the partial coloring procedure to the remaining free coordinates. Each iteration of partial coloring decreases the number of variables by half. The following lemma describes the guarantees of a partial coloring subroutine which we describe and analyze in the following section.

Lemma 26.2. *Let $S_1, \dots, S_m \subseteq [n]$, and let $x_0 \in [-1, 1]^n$. Let $\epsilon > 0$ be sufficiently small. Then there is a randomized algorithm that, in expected polynomial time, computes a point $x \in [-1, 1]^n$ such that*

(i) $|x(S_i) - x_0(S_i)| \leq O\left(\sqrt{n \log(m/n)}\right)$ for all j.

(ii) At least half of the coordinates are tight.

We now explain at a high level how the partial coloring bounds of Lemma 26.2 lead to a tight fractional covering with good discrepancy. Consider the first iteration. We apply `partial-coloring` and obtain a vector $x^{(1)}$ that (a) respects all of our desired discrepancy bounds and (b) makes at least half the variables tight. For all tight coordinates i, $x_i^{(1)}$ is forever fixed. Henceforth we restrict our attention to the remaining free variables $\mathcal{V}^{(1)}$, which has half as many variables as before. Consider now the second iteration. We apply `partial-coloring` in $\mathcal{V}^{(1)}$, now using $x^{(1)}$ (restricted to $\mathcal{V}^{(1)}$) as the initial point x_0. Now we obtain a fractional coloring $x^{(2)}$ over $\mathcal{V}^{(1)}$ where at least half the variables are tight, and where $x^{(2)} - x^{(1)}$ satisfies the desired discrepancy bound. This time, though, n is divided by half, which improves the discrepancy bound::

$$|x^{(2)}(S_i) - x^{(1)}(S_i)| \leq O\left(\sqrt{n \log(m/n)/2}\right).$$

Continuing in this fashion, we repeatedly color and remove half the free variables. Each iteration pays a geometrically decreasing cost in the discrepancy, so the overall discrepancy for each set S_i remains $O\left(\sqrt{n \log(m/n)}\right)$.

Pseuduocode for this iterative algorithm is given in Fig. 26.1. The following lemma and proof formalizes the high-level argument above.

```
iterative-partial-coloring($S_1, \dots, S_m$)
1. Let $x^{(0)} = \mathbb{0}$ and $y^{(0)} = \mathbb{0}$.
2. For $t = 1, 2, \dots$:
   A. Let $\mathcal{V}^{(t)} = \{i : y_i^{(t-1)} \in (1 - \epsilon, 1 + \epsilon)\}$.
   B. If $\mathcal{V}^{(t)}$ is empty then return $y^{(t-1)}$.
   /* Otherwise partially color the loose vertices $\mathcal{V}^{(t)}$ with starting point given by
      $y^{(t-1)}$ restricted to $\mathcal{V}^{(t)}$.                                              */
   C. Let $x^{(t)} = $ partial-coloring($S_1 \cap \mathcal{V}^{(t)}, \dots, S_m \cap \mathcal{V}^{(t)}$, $\mathcal{V}^{(t)}$),
   D. Set $y_i^{(t)} = y_i^{(t-1)}$ for $i \notin \mathcal{V}^{(t)}$ and $y_i^{(t)} = x_i^{(t)}$ for $i \in \mathcal{V}^{(t)}$.
```

Figure 26.1: Iteratively applying the `partial-coloring` procedure to obtain a tight fractional coloring. See Lemma 26.3.

Lemma 26.3. *In expected polynomial time,* `iterative-partial-coloring` *returns a fractional coloring $y \in [-1, 1]^n$ such that:*

(a) All coordinates y_j are tight.

(b) All sets S_i have $|y^{(t)}(S_i)| \leq O\Big(\sqrt{n \log(m/n)}\Big)$.

Proof. Let T be the (random) number of iterations completed by `iterative-partial-coloring`; that is, `iterative-partial-coloring` returns $y^{(T)}$.

For each iteration t, we have

$$|y^{(t)}(S_i)| \leq |x^{(t)}(S_i \cap \mathcal{V}^{(t)}) - y^{(t-1)}(S_i \cap \mathcal{V}^{(t)})| + |y^{(t-1)}(S_i)|,$$

hence

$$|y^{(T)}(S_i)| \leq \sum_{t=1}^{T} |x^{(t)}(S_i \cap \mathcal{V}^{(t)}) - y^{(t-1)}(S_i \cap \mathcal{V}^{(t)})|.$$

We also have, for each iteration t,

$$|x^{(t)}(S_i \cap \mathcal{V}^{(t)}) - y^{(t-1)}(S_i \cap \mathcal{V}^{(t)})| \leq O\Big(\sqrt{|\mathcal{V}^{(t)}| \log(m/|\mathcal{V}^{(t)}|)}\Big).$$

Recall that $|\mathcal{V}^{(t)}| \leq n/2^t$, and observe that

$$\frac{n}{2^t} \log\big(m/(n/2^t)\big) = \frac{n}{2^t} \log(m/n) + \frac{n}{2^t} t \leq O\Big(\frac{n}{1.9^t} \log(m/n)\Big).$$

Plugging back in we have

$$|y^{(T)}(S_i)| \leq O\Big(\sqrt{n\log(m/n)}\Big)\sum_{t=1}^{T} 1.9^{-t} = O\Big(\sqrt{n\log(m/n)}\Big),$$

as desired. □

Now we are prepared to prove Theorem 26.1, modulo the proof of Lemma 26.2 which we prove next.

Let $y \in [-1,1]^n$ be as described in Lemma 26.3. Let $z \in \{-1,1\}^n$ be the integral coloring obtained from y by deterministically setting $z_i \in \{-1,+1\}$ to the closer value to y_i. This increases the discrepancy by at most $\epsilon n \leq 1$ for each set S_i. This completes the proof.

26.3 Partial coloring by random walks: proof of Lemma 26.2

The partial coloring algorithm We now describe the algorithm underlying Lemma 26.2. Let $\lambda = c\sqrt{n\log(m/n)}$ for a sufficiently large constant c. Let

$$\mathcal{P} = \{x \in [-1,1]^n : |x(S_i) - x_0(S_i)| \leq \lambda \text{ for all } S_i\}.$$

Our goal is to find a fractional coloring $x \in \mathcal{P}$ where at least half the coordinates are tight.

We analyze an algorithm that can loosely be described as a high-dimensional random walk inside $\mathcal{P}$. The algorithm starts at $x^{(0)} = x_0$. Each iteration we move from $x^{(t-1)}$ to a random point $x^{(t)} \in \mathcal{P}$ where the step $x^{(t)} - x^{(t-1)}$ is sampled from a carefully chosen, high-dimensional Gaussian distribution.

As $x^{(t)}$ moves randomly through $\mathcal{P}$, $x^{(t)}$ may approach some of the boundaries of $\mathcal{P}$. For example, a coordinate $x_j^{(t)}$ may become tight, or the discrepancy of a set S_i may approach its limit λ_i. In this case the walk is constrained to avoid violating these critical constraints.

Towards a more formal description, let

$$\delta = \epsilon/\operatorname{poly}(m,n).$$

δ controls average step size of the random walk. Let $T = O(1/\delta^2) = \operatorname{poly}(m,n)/\epsilon$. The algorithm runs for T iterations. We want to show that at termination, $x^{(T)}$ is a tight coloring in $\mathcal{P}$.

After t iterations, we say that a *coordinate* $x_j^{(t)}$ *is tight* if $\left|x_j^{(t)}\right| \geq 1 - \epsilon$ (like before). We say that a *set* S_i *is tight* if

$$|\langle S_i, x^{(t)} - x_0 \rangle| \geq \lambda - \epsilon.$$

The next random step, $x^{(t+1)} - x^{(t)}$, is restricted to ensure that the tight coordinates and sets are not further violated, as follows.

Let $S^{(t)}$ be the subspace orthogonal to all tight sets and coordinates with respect to $x^{(t)}$:

$$S^{(t)} \overset{\text{def}}{=} \{y \in \mathbb{R}^n : y_j = 0 \text{ for all tight } x_j,\, y(S_i) = 0 \text{ for all tight } S_i\}.$$

Note that all the constraints are linear equations, so $S^{(t)}$ indeed describes a subspace. The step $x^{(t+1)} - x^{(t)}$ is sampled as a Gaussian vector from the subspace $S^{(t)}$.

To implement such a sample, let $k = \dim(S^{(t)})$, let $c_1, \ldots, c_k$ be an orthonormal basis for $S^{(t)}$, and let $U \in \mathbb{R}^{n \times k}$ be the matrix where the ith column is c_i. Let $g^{(t)} \sim \mathcal{N}^k$ be a vector of k independent, normal Gaussian variables. We set

$$x^{(t+1)} = x^{(t)} + \delta U g^{(t)}.$$

Observe that since $Ug^{(t)} \in S^{(t)}$, $x^{(t+1)}$ will have the same relationship with the variables and sets that were tight with respect to $x^{(t)}$; in particular, these tight inequalities will not get worse.

The following two lemmas help us understand the random vector $Ug^{(t)}$.

Lemma 26.4. *$\|Uw\|^2 = \|w\|^2$ for all $w \in \mathbb{R}^k$ and $\left\|U^\mathsf{T} v\right\|^2 \leq \|v\|^2$ for all $v \in \mathbb{R}^k$.*

Proof. We have

$$Uw = \sum_{i=1}^k w_i c_i.$$

Since $c_1, \ldots, c_k$ is an orthonormal basis, we have

$$\|Uw\|^2 = \left\langle \sum_{i=1}^k w_i c_i, \sum_{i=1}^k w_i c_i \right\rangle = \sum_{i=1}^k \sum_{j=1}^k w_i w_j \langle c_i, c_j \rangle = \sum_{i=1}^k w_i^2 = \|w_i\|^2.$$

For the second claim, observe

$$\left\|U^\mathsf{T} v\right\|^2 = \sum_{i=1}^k \langle c_1, v \rangle^2.$$

To interpret the RHS, extend $c_1, \ldots, c_k$ to an orthonormal basis $c_1, \ldots, c_n$ of $\mathbb{R}^n$. The vector $y \in \mathbb{R}^n$ defined by

$$y_i = \langle c_i, v \rangle \qquad\qquad i = 1, \ldots, n$$

is simply a change of coordinates of x to the new orthonormal basis, so

$$\|x\|^2 = \|y\|^2 = \sum_{i=1}^{n} \langle c_i, v \rangle^2.$$

Clearly the sum in the RHS is greater than the previous sum for $\left\|U^{\mathsf{T}} v\right\|^2$. □

Lemma 26.5. *Let $k = \dim(S^{(t)})$. For any vector v, $\langle v, Ug^{(t)} \rangle \sim \mathcal{N}(0, \sigma^2)$ where $\sigma^2 \leq \|v\|^2$.*

Proof. Let $b = U^{\mathsf{T}} v$. We have

$$\langle v, Ug^{(t)} \rangle = \left\langle U^{\mathsf{T}} v, g^{(t)} \right\rangle = \langle b, g^{(t)} \rangle.$$

As we know, $\langle b, g^{(t)} \rangle$ is distributed as a Gaussian with mean 0 and variance $\|b\|^2$. We have $\|b\|^2 = \left\|U^{\mathsf{T}} v\right\|^2 \leq \|v\|^2$ be Lemma 26.4. □

In particular, Lemma 26.5 implies that for each set i,

$$x^{(t+1)}(S_i) - x^{(t)}(S_i) = \delta \langle \mathbb{1}_{S_i}, Ug \rangle$$

is distributed as a Gaussian with mean 0 and variance at most

$$\delta^2 |S_i| \leq \delta^2 n.$$

(Here $\mathbb{1}_{S_i}$ denotes the $\{0,1\}$-indicator variable for S_i.) Lemma 26.5 also implies that each coordinate $j \in [n]$,

$$(x^{(t+1)} - x^{(t)})_j = \delta (Ug)_j$$

is distributed as a Gaussian with mean zero and variance at most δ^2. Now recall that for a Guassian random variable g with mean zero and variance σ^2,

$$\mathbf{P}[|g| \geq \lambda \sigma] \leq 2e^{-\lambda^2/2}$$

for all λ. Consequently we have

$$\mathbf{P}[|x^{(t+1)}(S_i) - x^{(t)}(S_i)| \geq \epsilon] \leq e^{-\epsilon^2/2\delta^2} = e^{-\operatorname{poly}(m,n)}$$

for each S_i and similarly

$$\mathbf{P}\left[\left|x_j^{(t+1)} - x_j^{(t)}\right| \geq \epsilon\right] \leq e^{-\operatorname{poly}(m,n)}$$

for each coordinate j. Meanwhile the quantities are always 0 for tight constraints and tight coordinates because $x^{(t+1)} - x^{(t)} \in S^{(t)}$.

Taking the union bound over all sets and variables, if $x^{(t)} \in \mathcal{P}$, then with high probability, we have $x^{(t+1)} \in \mathcal{P}$. Taking the union bound over all T iterations gives the following.

Lemma 26.6. *With high probability, $x^{(t)} \in \mathcal{P}$ for all $t \in [T]$.*

The remaining task is to show that at least half the variables are tight after T iterations.

Lemma 26.7. $\mathbf{E}\left[\|x^{(T)}\|^2\right] \geq T\,\mathbf{E}[\dim(S^{(T)})]$.

Proof. Each iteration t, we have

$$\|x^{(t)}\|^2 = \|x^{(t-1)}\|^2 + 2\langle x^{(t-1)}, \delta U^{(t)} g^{(t)}\rangle + \delta^2 \|U^{(t)} g^{(t)}\|^2.$$

When we take expectations, we have

$$\mathbf{E}[\langle x^{(t-1)}, \delta U^{(t)} g^{(t)}\rangle] = \delta\,\mathbf{E}\left[\left\langle (U^{(t)})^{\mathsf{T}} x^{(t-1)}, g^{(t)}\right\rangle\right] = 0$$

because $\left\langle (U^{(t)})^{\mathsf{T}} x^{(t-1)}, g^{(t)}\right\rangle$ is a mean 0 Gaussian. We also have

$$\mathbf{E}\left[\|U^{(t)} g^{(t)}\|^2\right] = \mathbf{E}\left[\|g^{(t)}\|^2\right] = \dim(S^{(t-1)}).$$

since $\|g^{(t)}\|^2$ is the sum of squares of $\dim(S^{(t)})$ standard Gaussians. Thus

$$\mathbf{E}\left[\|x^{(t)}\|^2 - \|x^{(t-1)}\|^2\right] = \delta^2\,\mathbf{E}[\dim(S^{(t-1)})].$$

Unrolling, we have

$$\mathbf{E}\left[\|x^{(T)}\|^2\right] \geq \sum_{t=1}^{T} \mathbf{E}\left[\|x^{(t)}\|^2 - \|x^{(t-1)}\|^2\right] \geq \sum_{t=1}^{T} \dim(S^{(t-1)}).$$

Finally, we observe that $\dim(S^{(t)}) \geq \dim(S^{(T)})$ since $S^{(t)}$ only shrinks over time with additional tight constraints and variables. □

Lemma 26.8. $\mathbf{E}\left[\begin{matrix}\text{\# tight coord.}\\ \text{w/r/t } x^{(T)}\end{matrix}\right] \geq \left(1 - \frac{1}{\delta^2 T}\right)n - \mathbf{E}\left[\begin{matrix}\text{\# tight sets}\\ \text{w/r/t } x^{(T)}\end{matrix}\right].$

Proof. We first observe that

$$\mathbf{E}\left[\|x^{(T)}\|^2\right] \geq \delta^2 T\,\mathbf{E}[\dim(S^{(T)})] \geq T\left(n - \mathbf{E}\begin{bmatrix}\text{\# tight sets}\\ \text{w/r/t } x^{(T)}\end{bmatrix} - \mathbf{E}\begin{bmatrix}\text{\# tight coord.}\\ \text{w/r/t } x^{(T)}\end{bmatrix}\right)$$

We also claim that $\mathbf{E}\left[\|x^{(T)}\|^2\right] \leq n$. If so, then we would have

$$n \geq \mathbf{E}\left[\|x^{(T)}\|^2\right] \geq \delta^2 T\left(n - \mathbf{E}\begin{bmatrix}\text{\# tight sets}\\ \text{w/r/t } x^{(T)}\end{bmatrix} - \mathbf{E}\begin{bmatrix}\text{\# tight coord.}\\ \text{w/r/t } x^{(T)}\end{bmatrix}\right),$$

which is the desired inequality up to rearrangement of terms.

To prove the claim, observe that for all coordinates j, and all iterations t, either the jth coordinate is tight, or

$$\mathbf{E}\left[\left(x_j^{(t)}\right)^2 - \left(x_j^{(t-1)}\right)^2 \,\middle|\, \left|x_j^{(t-1)}\right| < 1-\epsilon\right] \leq \delta^2.$$

It follows that $\mathbf{E}\left[\left(x_j^{(T)}\right)^2\right] \leq 1$ for all j, hence $\mathbf{E}\left[\|x^{(T)}\|^2\right] \leq n$. □

Lemma 26.9. $\mathbf{E}\begin{bmatrix}\text{\# tight sets}\\ \text{w/r/t } x^{(T)}\end{bmatrix} \leq \frac{n}{4}$, *hence* $\mathbf{E}\begin{bmatrix}\text{\# tight coord.}\\ \text{w/r/t } x^{(T)}\end{bmatrix} \geq \frac{3n}{4}$.

Proof. Fix a set S_i. Let $\mu = \lambda - \epsilon$; we again have $\mu \geq c_1\sqrt{n\log(m/n)}$ for an arbitrarily large constant c_1. We want to analyze the probability that $|x^{(T)}(S_i) - x^{(0)}(S_i)| \geq \mu$.

For $t \in [T]$, let

$$Y_t = x^{(t)}(S_i) - x^{(t-1)}(S_i) = \delta U^{(t)} g^{(t)}.$$

Observe that

$$x^{(T)}(S_i) - x^{(0)}(S_i) = Y_1 + \cdots + Y_T.$$

If they Y_t's were independent, then we know the sum behaves like a Guassian with variance at most $T\delta^2 n$.

Here the Y_t's are not independent: earlier iterations effect the variance of later iterations. However, each Y_t is a Gaussian conditional on $Y_1, \ldots, Y_{t-1}$. In this case, one can still show that their sum will still *behave* as if they were independent; that is, as a Gaussian with variance at most $T\delta^2 n$. (See Lemma 26.11 in Section 26.A.) Consequently

$$\mathbf{P}[Y_1 + \cdots + Y_T \geq \mu] \leq 2e^{-\frac{\mu^2}{2T\delta^2 n}} \leq 2e^{-\frac{c_1^2 n\log(m/n)}{T\delta^2 n}} \leq \frac{n}{8m}$$

for a sufficiently large constant c_1.

Now, by linearity of expectation, the expected number of tight sets is

$$\mathbf{E}\begin{bmatrix}\text{\# tight sets}\\ \text{w/r/t } x^{(T)}\end{bmatrix} = \sum_i \mathbf{P}[S_i \text{ is tight}] \leq m \cdot \frac{n}{8m} = \frac{n}{8},$$

as desired. □

We now complete the proof by showing that at least half the coordinates are tight with constant probability. The preceding lemma shows that the expected number of free coordinates is at most $n/4$. By Markov's inequality, there are at most $n/2$ free coordinates, hence at least $n/2$ tight coordinates, with probability at least $1/2$.

While we have only proven that `partial-coloring` succeeds with constant probability. Note that the guarantees of Lemma 26.2 is easy to verify. This leads to a Las Vegas algorithm with expected polynomial running time that repeats the `partial-coloring` algorithm until it succeeds.

26.4 Additional notes and materials

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

26.5 Exercises

Exercise 26.1. Show that a uniformly random coloring of the n points has maximum discrepancy at most $O\left(\sqrt{n \log m}\right)$ with high probability.

Exercise 26.2. Suppose every set contains at most L elements for some $L \in \mathbb{N}$. Adjust the $O\left(\sqrt{n \log(m/n)}\right)$-discrepancy algorithm of this chapter to obtain $O\left(\sqrt{L \log(m/n)} \log(n)\right)$ in this setting.[1] Here, rather than repeat the entire analysis verbatim, explain where L enters the analysis and how it fits in and propagates through the rest of the proof to obtain the desired bound. (At least one critical calculation should be redone.)

[1]In particular, show that Lemma 26.2 holds with discrepancy $O\left(\sqrt{L \log(m/n)}\right)$. The additional $\log(n)$ comes from running partial coloring $O(\log(n))$ times.

26.A Analyzing Gaussian martingales

The proof of Lemma 26.9 involves analyzing a sequence sum of random variables where each term, conditional on all previous terms, was distributed as a Gaussian. There we suggested that the sum behaves similarly as if it were a sum of independent Gaussians. Here we provide the formal details.

Lemma 26.10. *Let X be a Gaussian random variable with mean 0 and variance σ^2. Then for all $t \in \mathbb{R}$.*

$$\mathbf{E}\left[e^{tX}\right] = e^{t^2\sigma^2/2}.$$

Proof. WLOG we may assume $\sigma = 1$. Let $f(x) = e^{-x^2/2}$ denote the density function of X. We have

$$\mathbf{E}\left[e^{tX}\right] = \int e^{tx} f(x)\,dx \overset{\text{(a)}}{=} e^{t^2/2}\int f(x-t)\,dx = e^{t^2/2}\int f(x)\,dx = e^{t^2/2},$$

where (a) observes that

$$e^{tx}f(x) = \frac{1}{\sqrt{2\pi}}e^{tx-x^2/2} = \frac{e^{t^2/2}}{\sqrt{2\pi}}e^{-(x-t)^2/2} = e^{t^2/2}f(x-t).$$

□

Lemma 26.11. *Let $Y_1, \dots, Y_n \in \mathbb{R}$ be random variables where for each $i \in [n]$, conditional on $Y_1, \dots, Y_{i-1}$, Y_i is a Gaussian with mean 0 and variance at most σ^2. Then*

$$\mathbf{P}[Y_1 + \cdots + Y_n \geq \lambda] \leq e^{-\lambda^2/2n\sigma^2}.$$

Proof. Let $t \in \mathbb{R}$ be a parameter TBD. By exponentiating and taking Markov's inequality (as usual), we have

$$\mathbf{P}[Y_1 + \cdots + Y_n \geq \lambda] \leq \mathbf{E}\left[e^{tY_1+\cdots+Y_n}\right]e^{-t\lambda} = \prod_{i=1}^{n}\mathbf{E}\left[e^{tY_i}\,\middle|\,Y_1,\dots,Y_{i-1}\right]e^{-t\lambda}$$

For each i, by Lemma 26.10, we have

$$\mathbf{E}\left[e^{tY_i}\,\middle|\,Y_1,\dots,Y_{i-1}\right] \leq e^{t^2/2\sigma^2}.$$

Plugging back in, we have

$$\mathbf{P}[Y_1 + \cdots + Y_n \geq \lambda] \leq e^{nt^2\sigma^2/2-t\lambda}.$$

The RHS is minimized by $t = \lambda/n\sigma^2$, which gives

$$\mathbf{P}[Y_1 + \cdots + Y_n \geq \lambda] \leq e^{-\lambda^2/2n\sigma^2},$$

as desired. □

Chapter 27

Randomly Testing Boolean Functions

27.1 Testing boolean formulae with 3 queries, and 8 letter graph CSP's

A *boolean function* is a function $f : \{0,1\}^n \to \mathbb{R}$ that takes as input a sequence of bits and outputs a single value - often another bit.[1] Clearly *any* (deterministic) program making a binary decision is a boolean function, which makes boolean functions a natural object of study. Here we explore a *testing* approach that takes a boolean function f as a black box, queries f at a limited number of inputs, and analyzes the outputs to make useful statements about f.

The universality of boolean functions makes them attractive to study. But the same universality makes it seem rather daunting to be able to obtain concrete and useful observations about them. Nonetheless today we will see a few interesting things that one *can* do, at least *approximately*, by combination of *randomization* and an appropriate change of basis.

Today we will discuss a few introductory topics in *property testing*, which takes as input f and tries to decide if f has a certain property. We would only be able to do so approximately, and differentiate functions that have the property (exactly) from functions that fail to have the property for a constant fraction of the inputs. For example, we will show how to approximately test boolean functions for the following properties.

- *Linearity:* whether $f : \{0,1\}^n \to \{0,1\}$ satisfies $f(x+y) = f(x) + f(y)$ for all x and y.
- *Dictatorship:* whether $f : \{0,1\}^n \to \{-1,1\}$ is of the form $f(x) = (-1)^{x_i}$ for some $i \in [n]$.

[1] Of course one can consider functions that output more than one value - but real-valued boolean functions suffice for the current discussion. In fact this note only really requires boolean functions of the form $f : \{0,1\}^n \to \{-1,1\}$.

The main goal of today's discussion is to describe the following *universal tester* for proof systems.

Theorem 27.1. *Let $L \subseteq \{0,1\}^n$. Let $p = 2^{2^n}$. Then there is a randomized algorithm $A_L : \{0,1\}^n \times \{0,1\}^p \to \{0,1\}$ that, given oracle access to $x \in \{0,1\}^n$ and $y \in \{0,1\}^p$, has the following properties.*

(a) A_L makes 3 queries to bits of x or y.

(b) If $x \in L$, then there exists $y \in \{0,1\}^p$ such that $A_L(x,y) = 1$ always.

(c) If $x \notin L$, then for all $y \in \{0,1\}^p$, we have

$$\mathbf{P}[A_L(x,y) = 0] \geq .001 \min_{y \in L} \frac{\|x-y\|_0}{n}.$$

The connection between Theorem 27.1 to boolean functions is not self-evident. Let us briefly describe the algorithm underlying Theorem 27.1 at a high level, which will make the connection more clear. Let $N = 2^n$, and identify $\{0,1\}^n \equiv N$. For each $i \in [N]$, let $\chi_i : \{0,1\}^N \to \{-1,1\}$ be the function defined by

$$\chi_i(x) = \begin{cases} 1 & \text{if } x_i = 0 \\ -1 & \text{if } x_i = 1 \end{cases}$$

χ_i is called a *dictator function* and is the topic of Section 27.4. Identifying $L \subseteq \{0,1\}^n$ as a subset of $[N]$, let $D_L = \{\chi_i : i \in L\}$. Note that in general, Boolean functions $f : \{0,1\}^N \to \{-1,1\}$ can be expressed as $\left(2^N = 2^{2^n}\right)$-dimensional $\{-1,1\}$-vectors. We will create a test that, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^p$, simultaneously tests

(a) If $y \in D_L$.

(b) Conditional on $y \in D_L$, if $y = \chi_x$ (where we identify x with an index in N).

Thus the theorem reduces to understanding two tests about Boolean functions. We will build these tools over the course of this note and will finish with the proof of Theorem 27.1.

To motivate Theorem 27.1, recall that an important ingredient of the PCP theorem (Chapter 25) was a subroutine that took as input a constant-size boolean function and output a graph CSP that modeled them in an error-preserving fashion. Let us now show how to obtain this result using the universal tester above.

Theorem 27.2. *Given a language $L \subset \{0,1\}^n$, one can construct a graph-CSP with graph $G_L = (V_L, E_L)$, alphabet A, and constraints $\{C_e \subset A^2 : e \in E_L\}$ with the following properties.*

1. *$A = \{0, 1, \dots, 7\}$.*
2. *There are n vertices $X_1, \dots, X_n \in V_L$ that only take labels in $\{0,1\} \subset A$, and satisfy the following.*
    - *If $x_1, \dots, x_n \in \{0,1\}$ is such that $(x_1, \dots, x_n) \in L$, then there exists an assignment $\sigma : V_L \to \{0,1\}$ such that $\sigma(X_i) = x_i$ for all i and $\textsc{unsat}(\sigma \mid G_L) = 0$.*
    - *If $x_1, \dots, x_n \in \{0,1\}$ is such that $(x_1, \dots, x_n) \notin L$, then for all assignments $\sigma : V_L \to \{0,1\}$ such that $\sigma(X_i) = x_i$ for all i, we have*

$$\textsc{unsat}(\sigma \mid G_L) \geq .001 \min_{y \in L} \frac{\|x - y\|_0}{n}.$$

Proof. Fix a universal tester T for the language L that takes as input $x \in \{0,1\}^n$ and $y \in \{0,1\}^p$, for $p = 2^{2^n}$. We create a vertex for each of the following.

1. For each $i \in [n]$, a vertex X_i (modeling the input bit x_i).
2. For each $i \in [p]$, a vertex Y_i (modeling the proof bit y_i).
3. The universal tester T flips a finite number of coins. For every possible outcome of coin tosses ω, we create a vertex Z_ω.

We have an alphabet $A = \{0, \dots, 7\}$ which we identify with $\{0,1\}^3$. For every outcome of coin tosses ω, we create 4 constraints/edges involving Z_ω based on the mechanism of the tester T when the coin tosses are ω.

1. We create a self-loop at Z_ω where, given a label

$$\sigma(Z_\omega) = (\sigma_1(Z_\omega), \sigma_2(Z_\omega), \sigma_3(Z_\omega)) \in \{0,1\}^3,$$

   we satisfy the constraint iff the following conditions hold. When the coin tosses of T are ω, and the three queries return $\sigma_1(Z_\omega)$, $\sigma_2(Z_\omega)$, and $\sigma_3(Z_\omega)$, the tester accepts (x, y).

2. For $i = 1, 2, 3$, suppose the ith query of T is to the k_ith bit of x. Then we create a constraint between Z_ω and X_{k_i} that accepts iff $\sigma_i(Z_\omega) = \sigma(X_{k_i})$. Similarly, if instead the ith query is to k_ith bit of y, we make the same constraint between Z_ω and Y_k.

One can now verify that this graph CSP satisfies the conditions of the statement. In particular, because the universal tester has a rejection probability that is proportional to the distance from x to L, the fraction of unsatisfied constraints in any labeling extending x will be proportional to the distance from x to L. $\square$

Exercise 27.1. Complete the proof of Theorem 27.2 by verifying that the graph-CSP described above satisfies the claimed properties.

27.2 Fourier analysis of boolean functions

A *boolean function* is a real-valued function defined over bit-strings of a fixed length; i.e.,

$$f : \{0,1\}^n \to \mathbb{R}.$$

Note that sums of boolean functions and rescaled boolean functions are again boolean functions. In particular, we can identify the set of all boolean functions with the 2^n-dimensional Euclidean vector space $\mathbb{R}^{\{0,1\}^n}$. Here the ith coordinate of the "vector" f is the value $f(i)$. Rather than the standard Euclidean inner product $\langle x, y \rangle = \sum_i x_i y_i$, it is more convenient to rescale $\langle \cdot, \cdot \rangle$ to the following inner product that we denote $\langle \cdot, \cdot \rangle_{\mathrm{b}}$:

$$\langle f, g \rangle_{\mathrm{b}} \overset{\text{def}}{=} \frac{1}{2^n} \langle f, g \rangle = \mathop{\mathbf{E}}_{x \sim \{0,1\}^n} [f(x) g(x)],$$

where in the RHS x is sampled uniformly from $\{0,1\}^n$. Let

$$\|f\|_{\mathrm{b}} = \sqrt{\langle f, f \rangle_{\mathrm{b}}} = \sqrt{\mathop{\mathbf{E}}_{x}[f^2(x)]}$$

denote the corresponding norm. This norm rescales the standard Euclidean norm by $2^{-n/2}$. Here are a couple helpful identities to get us started.

Lemma 27.3. *For two boolean functions $f, g : \{0,1\}^n \to \{0,1\}$, we have*

$$\|f - g\|_{\mathrm{b}}^2 = \mathop{\mathbf{P}}_{x}[f(x) \neq g(x)].$$

Lemma 27.4. *For two boolean functions $f, g : \{0,1\}^n \to \{-1,1\}$, we have*

$$\mathbf{P}[f(x) \neq g(x)] = \frac{1}{4} \|f - g\|_{\mathrm{b}}^2.$$

Lemma 27.5. *For any boolean function $f : \{0,1\}^n \to \{-1,1\}$, we have $\|f\|_{\mathrm{b}}^2 = 1$.*

We leave the proofs of the above as exercises.

So far we have expressed boolean functions in terms of their "truth tables" as vectors in $\mathbb{R}^{\{0,1\}^n}$, but of course there are many possible bases over $\mathbb{R}^{\{0,1\}^n}$ that one could work with. *Fourier analysis* is based on the following choice of basis. For each set $S \subseteq [n]$, define a boolean function $\chi_S : \{0,1\}^n \to \{-1,1\}$ by

$$\chi_S(x) = (-1)^{\sum_{i\in S} x_i} = \begin{cases} 1 & \text{if } \sum_{i\in S} x_i \text{ is even,} \\ -1 & \text{if } \sum_{i\in S} x_i \text{ is odd.} \end{cases}$$

We call χ_S the *Sth Fourier basis function*; sometimes χ_S is called the parity function over S. The Fourier basis functions have many convenient properties of which we list a few. For $S = \emptyset$, we have $\chi_\emptyset = \mathbb{1}$, the all-one's vector. For all $S, T \subseteq [n]$, we have

$$\chi_S \chi_T = \chi_{(S \triangle T)}, \tag{27.1}$$

where $S \triangle T = (S \cup T) \setminus (S \cap T)$ denotes the symmetric difference. We also have, for all nonempty sets $S \neq \emptyset$,

$$\mathop{\mathbf{E}}_{x \in \{0,1\}^n}[\chi_S(x)] = 0.$$

The above is easy to see for singleton sets $S = \{i\}$. For general sets S, letting $i \in S$ and $S' = S - i$, we have

$$\mathbf{E}[\chi_S] \overset{\text{(a)}}{=} \mathbf{E}[\chi_{S'}(x)\chi_i(x)] \overset{\text{(b)}}{=} \mathbf{E}[\chi_{S'}(x)]\,\mathbf{E}[\chi_i(x)] \overset{\text{(c)}}{=} 0.$$

Here (a) is by (27.1). (b) is by independence. (c) is applies the singleton case. Finally, by combining the above observations, we have

$$\langle \chi_S, \chi_T \rangle_{\mathrm{b}} = \begin{cases} 1 & \text{if } S = T \\ 0 & \text{otherwise} \end{cases} \tag{27.2}$$

for any two sets $S, T \subset [n]$.

Exercise 27.2. Verify Eqs. (27.1) and (27.2) above.

Equation (27.2) means that the functions $\{\chi_S : S \subseteq [n]\}$ form an orthonormal set. There are also 2^n many of them, and we are working in a 2^n-dimensional space, so in fact they form an orthornomal basis (w/r/t $\langle \cdot, \cdot \rangle_{\mathrm{b}}$). Linear algebra then dictates that *any* boolean function $f : \{0,1\}^n \to \mathbb{R}$ can be written uniquely as a linear combination of the Fourier basis functions $\{\chi_S : S \subseteq [n]\}$, and this representation is given by

$$f = \sum_{S \subseteq [n]} \langle f, \chi_S \rangle_{\mathrm{b}} \chi_S = \sum_{S \subseteq [n]} \mathop{\mathbf{E}}_{x}[f(x)\chi_S(x)]\chi_S.$$

Let $\hat{f} : 2^n \to \mathbb{R}$ denote the coordinates in this basis; i.e., $\hat{f}_S = \langle f, \chi_S \rangle_{\mathrm{b}}$ for each set $S \subseteq [n]$. The map $f \mapsto \hat{f}$ is unitary with respect to the norms $\langle \cdot, \cdot \rangle_{\mathrm{b}}$ and $\langle \cdot, \cdot \rangle$; that is, a rotation that preserves distances. For any boolean functions $f, g : \{0,1\}^n \to \{0,1\}$ we have

$$\mathop{\mathbf{P}}_{x}[f(x) \neq g(x)] = \langle f - g, f - g \rangle_{\mathrm{b}} = \langle \hat{f} - \hat{g}, \hat{f} - \hat{g} \rangle = \|\hat{f} - \hat{g}\|^2,$$

where $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ are the standard Euclidean norm. One should not underestimate the significance of this transformation. The Fourier transform gives a unitary transformation that maps boolean functions $f : \{0,1\}^n \to \{0,1\}$ into a Euclidean vector space such that the probability of two functions agreeing is captured exactly by the norm.

27.3 Linearity

A boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to be *linear (mod 2)* if

$$f(x + y) = f(x) + f(y)$$

for all $x, y \in \{0,1\}^n$, where all additions are made modulo 2. For technical reasons it is instead convenient to consider functions of the form $f : \{0,1\}^n \to \{-1,1\}$, and define such a function to be linear if

$$f(x + y) = f(x)f(y)$$

for all $x, y \in \{0,1\}^n$. Of course, by mapping 0 to 1 and 1 to -1, we can easily convert one type of linear function to the other. Note that the Fourier basis functions $\chi_S : \{0,1\}^n \to \{-1,1\}$ are linear functions in the sense immediately above.

27.3.1 Testing linearity

Our goal is to devise an algorithm that, given a boolean function $f : \{0,1\}^n \to \{-1,1\}$, decides if f is a linear function. Of course we can query f everywhere but we would prefer to test f with only a few queries. We point out that a deterministic and exact algorithm is impossible with only a few queries, but still we will be able to show some interesting approximate and randomized guarantees.

The following simple procedure is maybe the most obvious one to try.

1. Draw $x, y \in \{0,1\}^n$ independently and uniformly at random.
2. Evaluate $f(x)$, $f(y)$, and $f(x+y)$.
3. Accept f if $f(x+y) = f(x)f(y)$.

This algorithm was analyzed by [BLR93b], who proved the following.

Theorem 27.6. *Let $f : \{0,1\}^n \to \{-1,1\}$. Then*

$$\mathop{\mathbf{P}}_{x,y}[f(x)f(y) \neq f(x+y)] \geq \min_S \mathop{\mathbf{P}}_x[f(x) \neq \chi_S(x)]$$

where $x, y \in \{0,1\}^n$ are distributed uniformly and independently over $\{0,1\}^n$.

Proof. For ease of notation, let $P = \mathbf{P}_{x,y}[f(x)f(y) = f(x+y)]$. Let $Z \in \{0,1\}$ be the indicator variable for the event that $f(x+y) = f(x)f(y)$. We have $P = \mathbf{E}[Z]$. We also have

$$\begin{aligned} Z &= 1 - \frac{1}{4}(f(x)f(y) - f(x+y))^2 \\ &\overset{\text{(a)}}{=} 1 - \frac{1}{4}(2 - 2f(x)f(y)f(x+y)) \\ &= \frac{1}{2}(1 + f(x)f(y)f(x+y)), \end{aligned}$$

where (a) observes that $f^2(x) = f^2(y) = f^2(x+y) = 1$. Thus

$$P = \mathbf{E}[Z] = \frac{1}{2} + \frac{1}{2}\mathop{\mathbf{E}}_x\left[f(x)\mathop{\mathbf{E}}_y[f(y)f(x+y)]\right] = \frac{1}{2} + \frac{1}{2}\langle f, h\rangle_{\text{b}} \overset{\text{(b)}}{=} \frac{1}{2} + \frac{1}{2}\langle \hat{f}, \hat{h}\rangle,$$

where we define $h(x) = \mathbf{E}_y[f(y)f(x+y)]$. (b) applies the Fourier transform. *We claim that $\hat{h}_S = \hat{f}_S^2$ for all S.* Indeed, we have

$$\begin{aligned} \mathop{\mathbf{E}}_x[h(x)\chi_S(x)] &= \mathop{\mathbf{E}}_{x,y}[f(y)f(x+y)\chi_S(x)] \overset{\text{(c)}}{=} \mathop{\mathbf{E}}_{x,y}[f(y)f(x+y)\chi_S(y)\chi_S(x+y)] \\ &= \mathop{\mathbf{E}}_y[f(y)\chi_S(y)]\mathop{\mathbf{E}}_{x,y}[f(x+y)\chi_S(x+y)] = \langle f, \chi_S\rangle_{\text{b}}^2 = \hat{f}_S^2. \end{aligned}$$

(c) is by linearity of χ_S. (d) observes that y and $x+y$ are independently and uniformly distributed in $\{0,1\}^n$. Plugging back in, we now have

$$P = \frac{1}{2} + \frac{1}{2}\sum_S \hat{f}_S^3 \overset{\text{(d)}}{\leq} \frac{1}{2} + \frac{1}{2}\max_S \hat{f}_S$$

(e) applies the fact that $\|\hat{f}\|^2 = 1$ for $f : \{0,1\} \to \{-1,1\}$. Rearranging, we have

$$\hat{f}_S \geq 2P - 1$$

for some set $S \subseteq [n]$. But then

$$4 \underset{x}{\mathbf{P}}[f(x) \neq \chi_S] = \|f - \chi_S\|_{\mathrm{b}}^2 = \|f\|_{\mathrm{b}} + \|\chi_S\|_{\mathrm{b}} - 2\langle f, \chi_S \rangle = 2 - 2\hat{f}_S \leq 4(1 - P),$$

as desired. □

If f is linear, then the linearity test succeeds one hundred percent of the time. But then the above theorem asserts there exists a basis function χ_S that agrees with f one hundred percent of the time. That is:

Corollary 27.7. *All linear functions $f : \{0,1\}^n \to \{-1,1\}$ are of the form χ_S for some set S.*

27.3.2 Locally correcting for linearity

Now suppose $f : \{0,1\} \to \{-1,1\}$ is ϵ-close to a basis function χ_S for an unknown set S. That is, $f(x) = \chi_S(x)$ for $(1-\epsilon)$-fraction of choices of x.

Can we recover S from f? The following theorem gives a randomized algorithm that uses f to simulate $\chi_S(x)$ for any x, even when $f(x) \neq \chi_S(x)$.

Theorem 27.8. *Let $f : \{0,1\} \to \{-1,1\}$ be ϵ-close to a basis function $\chi_S : \{0,1\} \to \{-1,1\}$. Given $x \in \{0,1\}^n$, consider the random value $f(x)f(x+y) \in \{-1,1\}$ where $y \in \{0,1\}^n$ is sampled uniformly at random. Then*

$$\underset{y}{\mathbf{P}}[f(y)f(x+y) = \chi_S(x)] \geq 1 - 2\epsilon.$$

Proof. $x + y$ and y are both distributed uniformly over $\{0,1\}^n$, and we have $f(y) = \chi_S(y)$ and $f(x+y) = \chi_S(x+y)$ each with probability of error $\leq \epsilon$. By the union bound, both occur with probability of error $\leq 2\epsilon$. But then we recover $\chi_S(x) = \chi_S(x+y)\chi_S(y)$. □

27.3.3 A remark on convolutions

A key component of the proof of Theorem 27.6 is the identity $\hat{h}_S = \hat{f}_S^2$ for the function $h(x) = \mathbf{E}_y[f(y)f(x+y)]$. More generally, for two boolean formulas $f, g : \{0,1\} \to \mathbb{R}$, the *convolution* of f and g, denoted $f * g$, is the function defined by

$$(f * g)(x) = \underset{y}{\mathbf{E}}[f(x)g(x+y)].$$

The following identity is called *Plancheral's identity* and generalizes the calculations used in Theorem 27.6.

Lemma 27.9. *Let $f, g : \{0,1\} \to \mathbb{R}$. Then $\widehat{(f * g)}_S = \hat{f}_S \hat{g}_S$ for all $S \subseteq [n]$.*

Exercise 27.3. Prove Lemma 27.9.

27.4 Dictators

A function $f : \{0,1\}^n \to \{-1,1\}$ is a *dictator* if it is one of the singleton basis functions[2],

$$f = \chi_i \text{ for some } i \in [n].$$

We want to test if a function $f : \{0,1\}^n \to \{-1,1\}$ is a dictator. We do so by combining two tests. First, clearly, any dictator function is linear, which gives our first test.

1. *Linearity test:* Sample $x, y \in \{0,1\}^n$ independently and reject if $f(x+y) \neq f(x)f(y)$.

The second test is new. Let $\Omega = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ be the set of triplets where not all coordinates are equal. Abusing notation, we write Ω^n to denote the triplets of vectors $x, y, z \in \{0,1\}^n$ such that for all i, $(x_i, y_i, z_i) \in \Omega$. To sample a uniformly random $(x,y,z) \in \Omega$, one can independently sample, for each $i \in [n]$, three coordinates $(x_i, y_i, z_i) \in \Omega$ uniformly at random. We write $(x,y,z) \sim \Omega^n$ when $(x,y,z) \in \Omega^n$ is sampled uniformly at random.

For any dictator function $f = \chi_i$, and $(x,y,z) \in \Omega^n$, we have $(f(x), f(y), f(z)) \in \Omega$. This motivates our second test:

2. *Not-all-equal (NAE) test:* Sample $x, y, z \sim \Omega^n$. Reject f unless $(f(x), f(y), f(z)) \in \Omega$.

Theorem 27.10. *Let $f : \{0,1\}^n \to \{-1,1\}$ be a boolean function. Suppose f passes both the linearity and not-all-equals test with probability $1-\epsilon$ for $\epsilon \leq .1$. Then there exists a coordinate $i \in [n]$ such that*

$$\mathop{\mathbf{P}}_{x}[f(x) \neq \chi_i(x)] \leq \epsilon,$$

where $x \in \{0,1\}^n$ is sampled uniformly at random.

To prove Theorem 27.10, we first require the following lemma analyzing the not-all-equal test.

Lemma 27.11. *Let $f : \{0,1\}^n \to \{-1,1\}$. Let $(x,y,z) \sim \Omega^n$. Then*

$$\mathbf{P}[(f(x), f(y), f(z)) \in \Omega] \leq \frac{7}{9} + \frac{2}{9}\sum_{i=1}^{n} \hat{f}_i^2.$$

[2]For ease of notation, we write χ_i instead of $\chi_{\{i\}}$.

We will prove this lemma below in Section 27.4.3. First, let us use it to prove Theorem 27.10.

Proof of Theorem 27.10. By the NAE test, we have $\sum_i \hat{f}_i^2 \geq 1 - 4.5\epsilon$. By the linearity test, we have that $\hat{f}_S \geq 1 - 2\epsilon$ for some S. But this set S must be a singleton $\{i\}$ because otherwise we have

$$1 = \|\hat{f}\|^2 \geq 1 - 4.5\epsilon + (1 - 2\epsilon)^2 > 1,$$

a contradiction. Thus $\hat{f}_i \geq 1 - 2\epsilon$ for some i. Then

$$4 \underset{x}{\mathbf{P}}[f(x) \neq \chi_i(x)] \overset{\text{(a)}}{=} \|f - \chi_i\|_{\text{b}}^2 = \|\hat{f} - \hat{\chi_i}\|^2 \overset{\text{(b)}}{=} (\hat{f}_i - 1)^2 + 1 - \hat{f}_i^2 = 2 - 2\hat{f}_i \leq 4\epsilon,$$

as desired. (a) is by Lemma 27.4. (b) takes the Fourier transform. (c) uses the identity $\|\hat{f}\|^2 = 1$ for all $f : \{0,1\}^n \to \{-1,1\}$. ☐

The dictatorship test we have just developed requires 6 queries to f: three for the linearity test, and three for the not-all-equals test. We can reduce this to three queries at the cost of increase the error rate with a simple trick, as follows.

Theorem 27.12. *Let $n \in \mathbb{N}$. There is a 3-query test for the family of dictators $D = \{\chi_i \mid i \in [n]\}$ with the following guarantee. Given a function $f : \{0,1\}^n \to \{-1,1\}$:*

1. *If $f \in D$, then the test always accepts f.*
2. *If f is ϵ-far from any dictator function and $\epsilon \leq .2$, then the test f with probability $\geq \epsilon/4$.*

Proof. We choose *either* the linearity test or not-all-equals test, randomly selecting one of the two with equal probability. Clearly, if f is a dictator, then the test always passes. Otherwise, suppose f fails the test with probability $\leq p$. Consider the test where we run *both* tests on f; f fails this test with probability $\leq 2p$. It follows that for $p \leq .05$, f is at most $4p$-far from some dictator. ☐

27.4.1 Families of dictators

We can extend the dictator test above to subfamilies of dictator functions as follows. For any set $S \subset [n]$, let

$$D_S = \{\chi_i : \{0,1\}^n \to \{-1,1\} \mid i \in S\}$$

be the set of dictator functions for coordinates $i \in S$. Suppose that given $f : \{0,1\}^n \to \mathbb{R}$ and $S \subset [n]$, we want to test if f is close to any dictator function in S. Consider the following.

1. With probability 1/2, run the dictatorship test from Theorem 27.10.
2. With probability 1/2, run the locally correcting protocol for linear functions for f with input string $\mathbb{1}_S$, accepting f if this protocol returns 1.

This test has the following bounds.

Theorem 27.13. *Given $S \subseteq [n]$, there is a 3-query test for the subfamily of dictators D_S with the following guarantee. Given a function $f : \{0,1\}^n \to \{-1,1\}$:*

1. *If $f \in D_S$, then the test always accepts f.*
2. *If f is ϵ-far from any function in D_S, then the test rejects f with probablity $\geq c\epsilon$ for some universal constant $c > 0$.*

Proof. The first property is immediate. Suppose $f \notin D_S$ and fails the test with probability p. Then f fails either test with probability $\leq 2p$. The first test implies that f is (cp)-far from a dictator for some universal constant $c > 0$. Because f is (cp)-far from a dictator χ_i and in particular from a linear function, the correction protocol returns $\chi_i(\mathbb{1}_S)$ with probability of error $\leq dcp$ for a universal constant $d > 0$. Since f passes that test with probability $2p$, we conclude that f is $O(p)$ close to χ_i for some $i \in \chi_i$. □

27.4.2 Noisy perturbation of boolean functions

It remains to analyze the not-all-equal test. Doing so requires analyzing boolean functions under random perturbations of their input, as follows.

For $x \in \{0,1\}^n$ and $p \in [0,1]$, let $\mathcal{N}_p(x)$ be the distribution of random strings where each bit x_i is flipped independently with probability p. The random function $\mathcal{N}_p$ arose previously in the analysis of error correcting codes. For a boolean function $f : \{0,1\}^n \to \mathbb{R}$, we define the boolean function $\mathrm{T}_p f : \{0,1\}^n \to \mathbb{R}$ by

$$(\mathrm{T}_p f)(x) = \mathop{\mathbf{E}}_{y \sim \mathcal{N}_p(x)}[f(y)].$$

Lemma 27.14. *Let $f : \{0,1\} \to \mathbb{R}$ be a boolean function. For $S \subset [n]$, $\widehat{(\mathrm{T}_p f)}_S = (1-2p)^{|S|}\hat{f}_S$.*

Proof. Since T_p and taking the Fourier transform are both linear functions, it suffices to prove the claim for $f = \chi_S$. We have

$$\begin{aligned}(\mathrm{T}_p \chi_S)(x) &= \mathop{\mathbf{E}}_{y \sim \mathcal{N}_p(x)}[\chi_S(y)] = \prod_{i \in S} \mathop{\mathbf{E}}_{y \sim \mathcal{N}_p(x)}[(-1)^{y_i}] = \prod_{i \in S}((1-p)(-1)^{x_i} - p(-1)^{x_i}) \\ &= \prod_{i \in S}(1-2p)(-1)^{x_i} = (1-2p)^{|S|}\chi_S(x),\end{aligned}$$

as desired. $\square$

27.4.3 Analysis of the not-all-equals test

Finally, let us analyze the not-all-equals test and prove Lemma 27.11.

Lemma 27.11. *Let $f : \{0,1\}^n \to \{-1,1\}$. Let $(x,y,z) \sim \Omega^n$. Then*

$$\mathbf{P}[(f(x), f(y), f(z)) \in \Omega] \le \frac{7}{9} + \frac{2}{9}\sum_{i=1}^{n} \hat{f}_i^2.$$

Proof. Let $P = \mathbf{P}[(f(x), f(y), f(z)) \in \Omega]$. Define a boolean function $\mathrm{NAE} : \{-1,1\}^3 \to \{0,1\}$ by setting

$$\mathrm{NAE}(a,b,c) = \begin{cases} 0 & \text{if } a = b = c, \\ 1 & \text{otherwise.} \end{cases}$$

We have

$$\begin{aligned}
\mathrm{NAE}(a,b,c) &= \frac{1}{8}\Big((a-b)^2 + (a-c)^2 + (b-c)^2\Big) \\
&= \frac{1}{8}\Big(2a^2 + 2b^2 + 2c^2 - 2ab - 2ac - 2bc\Big) \\
&= \frac{3}{4} - \frac{1}{4}(ab + ac + bc).
\end{aligned}$$

Thus

$$\begin{aligned}
P &= \mathbf{E}[\mathrm{NAE}(f(x), f(y), f(z))] = \frac{3}{4} - \frac{1}{4}\mathbf{E}[f(x)f(y) + f(y)f(z) + f(x)f(z)] \\
&\overset{\text{(a)}}{=} \frac{3}{4} - \frac{3}{4}\mathbf{E}[f(x)f(y)]
\end{aligned}$$

where (a) is by symmetry of Ω. Consider $\mathbf{E}[f(x)f(y)]$. We have

$$\begin{aligned}
\mathbf{E}[f(x)f(y)] &\overset{\text{(b)}}{=} \mathop{\mathbf{E}}_{x,y \sim \mathcal{N}_{2/3}(x)}[f(x)f(y)] = \Big\langle f, \mathrm{T}_{2/3}\, f \Big\rangle_{\mathrm{b}} \overset{\text{(c)}}{=} \langle \hat{f}, \widehat{\Big(\mathrm{T}_{2/3}\, f\Big)} \rangle \\
&= \sum_S (-1/3)^{|S|} \hat{f}_S^2 \ge -\frac{1}{3}\sum_i \hat{f}_i^2 - \frac{1}{27}\sum_{\substack{|S| \ge 3 \\ |S| \text{ odd}}} \hat{f}_S^2 \\
&\overset{\text{(d)}}{\ge} -\frac{1}{3}\sum_i \hat{f}_i^2 - \frac{1}{27}\left(1 - \sum_i \hat{f}_i^2\right) = -\frac{1}{27} - \frac{8}{27}\sum_i \hat{f}_i^2.
\end{aligned}$$

Here (b) observes that x is sampled uniformly from $\{0,1\}^n$, and conditional on x, y is distributed as $\mathcal{N}_{2/3}(x)$. (c) applies the unitary Fourier transform. (d) is by Lemma 27.14. (e) is because $\sum_S \hat{f}_S^2 = \|\hat{f}\|^2 = 1$. Plugging back in, we have

$$P \leq \frac{3}{4} - \frac{3}{4}\left(-\frac{1}{27} - \frac{8}{27}\sum_i \hat{f}_i^2\right) = \frac{7}{9} + \frac{2}{9}\sum_i \hat{f}_i^2,$$

as desired. □

27.5 Universal Tester

We have arrived at the final section of this note, where we use our newly developed toolkit for analyzing Boolean function to analyze the universal tester introduced in Section 27.1. We restate the claim for the reader's convenience.

Theorem 27.1. *Let $L \subseteq \{0,1\}^n$. Let $p = 2^{2^n}$. Then there is a randomized algorithm $A_L : \{0,1\}^n \times \{0,1\}^p \to \{0,1\}$ that, given oracle access to $x \in \{0,1\}^n$ and $y \in \{0,1\}^p$, has the following properties.*

(a) A_L makes 3 queries to bits of x or y.

(b) If $x \in L$, then there exists $y \in \{0,1\}^p$ such that $A_L(x,y) = 1$ always.

(c) If $x \notin L$, then for all $y \in \{0,1\}^p$, we have

$$\mathbf{P}[A_L(x,y) = 0] \geq .001 \min_{y \in L} \frac{\|x - y\|_0}{n}.$$

Proof. We identify the input space $\{0,1\}^n$ with the family of all integers $[N]$ where $N = 2^n$, and the language L as a subset of indices from $[N]$. The proof space $\{0,1\}^p$, where $p = 2^N$, consists of the family of all boolean functions $f : \{0,1\}^N \to \{-1,1\}$. A 1-bit query to a function f evaluates $f(x)$ for a bit string $x \in \{0,1\}^n$.

For $i \in L$, the proof that $i \in L$ is the dictator function χ_i. We need to create a randomized algorithm $A_L(i,f)$ such that:

(a) A_L makes at most 3 queries to i or f.

(b) If $i \in L$, there $A_L(i,\chi_i) = \texttt{true}$ always.

(c) If $i \notin L$, then for all $f : \{0,1\}^n \to \{-1,1\}$,

$$\mathbf{P}[A_L(i,f) = \texttt{false}] \geq .001 \min_{j \in L} \frac{\|i - j\|_0}{n}.$$

Our protocol will consist of two sets.

1. We test that $f = \chi_j$ for some $j \in L$, using the test from Theorem 27.13.
2. Given that $f = \chi_j$ for some $j \in L$, we test if $i = j$.

For the first test, we recall the guarantees from Theorem 27.13. The test takes 3 queries. If $f = \chi_j$ for some $j \in L$, it always accepts f. Otherwise, if f is ϵ-far from L, then it rejects j with probability $C_1\epsilon$ for some universal constant $C_1 > 0$.

The second step is based on the correction protocol from Section 27.3.2. Suppose $f \approx \chi_j$ for some $j \in L$. We need to determine if $i = j$, rejecting with probability proportional to their Hamming distance as bit-strings.

Ideally we could sample an index $\tau \in [n]$ and accept iff $i_\tau = j_\tau$. The catch is that we don't know j. However, we can still (approximately) query χ_j via f and the linear correction protocol.

To build some intuition, suppose $f = \chi_j$ (but we don't know j). For fixed τ, we need an input $x \in \{0,1\}^N$ such that

$$f(x) = \chi_j(x) = j_\tau$$

for all τ. Then we could compare $f(x)$ to i_τ. To this end, define $x \in \{0,1\}^N$ by

$$x_k = k_\tau$$

for all $k \in [N]$, where k_τ refers to the τth bit of k. Then $\chi_j(x) = j_\tau$ for all j.

Suppose $i \in L$. Let $f = \chi_i$. Then f passes the first test, since $f \in D_L$. Then it passes the next test, since for any fixed τ, $f(x) = \chi_i(x) = i_\tau$.

Now suppose $i \notin L$, and has Hamming distance ϵ from any $j \in L$. The proof f is either ϵ-far from any dictator in D_L, or ϵ-close to some dictator in D_L.

If f is ϵ-far from any dictator in D_L, then the first test rejects f is probability $\Omega(\epsilon)$.

Otherwise f is ϵ-close to χ_j for some $j \in L$. The second test samples a uniformly random index $\tau \in [n]$. Since $j \in L$, j is ϵ-far from i, so $j_\tau \neq i_\tau$ with probability ϵ. Let $x \in \{0,1\}^N$ be the test input described above for this choice of τ. The linear correction protocol outputs

$$\chi_j(x) = j_\tau$$

with probability $1 - 2\epsilon$. So with probability $\epsilon(1 - 2\epsilon)$, the second test fails.

So in either case, we reject (i, f) with probability $\Omega(\epsilon)$. $\square$

27.6 Additional notes and materials

This chapter is motivated by the PCP theorem and limited in scope to the techniques that lead to the universal tester. There are many other applications of property testing and Boolean analysis. See [ODo14] for a booklength treatment on these topics. These notes are based on chapters 1, 2, and 7 of [ODo14]. Property testing also extends beyond boolean functions. We recommend Prof. Grigorescu's Spring 2021 class on *sublinear time algorithms* for more topics in this area.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

27.7 Exercises

Exercise 27.1. Complete the proof of Theorem 27.2 by verifying that the graph-CSP described above satisfies the claimed properties.

Exercise 27.2. Verify Eqs. (27.1) and (27.2) on page 356.

Exercise 27.3. Prove Lemma 27.9.

Chapter 28

Sparsification

Generally speaking, *sparsfication* refers to the idea of taking one complex object and replacing it with a smaller and simpler one that preserves certain salient properties. For example, geometric ϵ-samples replace arbitrarily large point sets with small, reweighted point sets while approximately preserving the measure of every range.

This chapter introduces two more examples of sparsification. The first is graph sparsification, where we reduce the number of edges in an undirected graph while approximately preserving all cuts. The second is matrix sparsification, where we sparsify sums of positive semi-definite matrices while preserving all of its spectral properties. The latter generalizes the former via the Laplacian. Due to time constraints we will not prove the sparsification results, but instead focus on some of their applications.

28.1 Graph sparsification

Let $G = (V, E)$ be an undirected graph with m

Theorem 28.1 ([BK15]). *Let $G = (V, E)$ be an undirected graph, m edges, n vertices, and positive edge weights $w : E \to \mathbb{R}_{\geq 0}$. Then there exists a subgraph $\tilde{G} = (V, E')$ (where $E' \subseteq E$) and weights $\tilde{w} : E' \to \mathbb{R}_{\geq 0}$ such that:*

(a) $\tilde{G}$ has $\left|\tilde{E}\right| \leq O\left(n \log(n)/\epsilon^2\right)$ edges.

(b)

preserves the weight of all cuts of G up to a $(1+\epsilon)$-factor. That is, for all $S \subset V$,

$$(1-\epsilon)\tilde{w}(\delta(S)) \leq w(\delta(S)) \leq (1+\epsilon)\tilde{w}(\delta(S))$$

Moreover, $(\tilde{G}, \tilde{w})$

Due to time constraints we will note prove Theorem 28.1. Instead we will show something weaker that still sheds some insight into Theorem 28.1: we will compute a sparse reweighted subgraph where the minimum cut is preserved up to a $(1 \pm \epsilon)$-factor.

Let λ denote the weight of the minimum cut in G. Let $\tau = \frac{\epsilon^2 \lambda}{c \log n}$ for a sufficiently large constant $c > 0$. Consider the randomized weights $\tilde{w}$ where for each edge $e \in E$, independently, we have

$$\tilde{w}(e) = \tau \cdot \begin{cases} \left\lceil \frac{w(e)}{\tau} \right\rceil & \text{with probability } p_e \stackrel{\text{def}}{=} \frac{w(e)}{\tau} - \left\lfloor \frac{w(e)}{\tau} \right\rfloor \\ \left\lfloor \frac{w(e)}{\tau} \right\rfloor & \text{with probability } 1 - p_e. \end{cases}$$

Equivalently, $\tilde{w}(e)$ is defined by

$$\tilde{w}(e) \in \left\{ \left\lfloor \frac{w(e)}{\tau} \right\rfloor \tau, \left\lceil \frac{w(e)}{\tau} \right\rceil \tau \right\} \text{ and } \mathbf{E}[\tilde{w}(e)] = w(e).$$

Consider a cut $\delta(S)$, where $S \subsetneq V$. The randomized weight of $\delta(S)$,

$$\tilde{w}(\delta(S)) = \sum_{e \in \delta(S)} \tilde{w}(e),$$

is a sum in independent random variables where the random part of each random variable varies by at most τ. The expected value is

$$\mathbf{E}[\tilde{w}(\delta(S))] = w(\delta(S)).$$

Since $\tau \leq (\epsilon^2 / c \log n) w(\delta(S))$ for a large constant c, by the multiplicative Chernoff bound, we have

$$|\tilde{w}(\delta(S)) - w(\delta(S))| \leq \epsilon w(\delta(S))$$

with high probability.

In particular, if $\delta(S)$ is the minimum cut, then $\tilde{w}(\delta(S)) \leq (1 + \epsilon)\lambda$ with high probability. Thus the minimum cut with respect to $\tilde{w}$ is at most $(1 + \epsilon)\lambda$ w/h/p.

Next we want to show that the minimum weight is at least $(1 - \epsilon)\lambda$ with high probability. Call a cut $\delta(S)$ *bad* if

$$\tilde{w}(\delta(S)) \leq (1 - \epsilon)\lambda,$$

and *good* otherwise. While any individual cut is good w/h/p, we cannot simply take a union bound over all cuts because they are exponentially many cuts.

Recall that there are at most $\binom{n}{2}$ minimum cuts in the graph (Chapter 4). An extension of the same argument shows that there are at most n^{2t} cuts of weight at most $t\lambda$, for all $t \in \mathbb{N}$ (exercise C.22).

Now, for $t \in \mathbb{N}$, let Q_t be the family of edge cuts with weight in the range $[t\lambda, (t + 1)\lambda)$...

28.2 Spectral sparsification

Recall that a matrix A is *positive semi-definite (PSD)* if it is symmetric ($\langle x, Ay\rangle = \langle y, Axy\rangle$ for all x, y) and $\langle x, Ax\rangle \geq 0$ for all x. For example, the Laplacian L_e of an edge $e = \{u, v\}$, Defined by $\langle x, L_e x\rangle = (x_u - x_v)^2$ for $x \in \mathbb{R}^V$, is positive semi-definite. So is the Laplacian $L = \sum_{e \in E} w(e) L_e$ of an undirected graph $G = (V, E)$ with edge weights $w \in \mathbb{R}^E_{>0}$.

Let A, B be two PSD matrices. We write

$$A \preceq B \text{ if } \langle x, Ax\rangle \leq \langle x, Bx\rangle \text{ for all } x \in \mathbb{R}^V.$$

$\preceq$ defines a partial order on the family of PSD matrices (over a fixed vector space). We say that B is a $(1 \pm \epsilon)$-approximation of A if

$$(1 - \epsilon)B \preceq A \preceq (1 + \epsilon)B.$$

If B is an $(1 \pm \epsilon)$-approximation of A then many properties of B are within a $(1 \pm \epsilon)$-factor of A. For examples, the eigenvalues of B match those of A up to a $(1 \pm \epsilon)$-factor:

Exercise 28.1. Suppose B is an $(1 \pm \epsilon)$-approximation of A. Prove that for all i, if $\mu_{A,i}$ is the ith largest eigenvalue of A, and $\mu_{B,i}$ is the ith largest eigenvalue of B, then

$$(1 - \epsilon)\mu_{B,i} \leq \mu_{A,i} \leq (1 + \epsilon)\mu_{B,i}.$$

Let $A = \sum_{i=1}^m B_i$ be a sum of m PSD matrices. We are interested in computing a *sparse sum* $\tilde{A} = \sum_{i=1}^m w_i B_i$, where $w_i \geq 0$ for all i, such that

(a) $(1 - \epsilon)\tilde{A} \preceq A \preceq (1 + \epsilon)\tilde{A}$.

(b) $w_i > 0$ for as few indices i as possible.

Such an $\tilde{A}$ would be useful as it could replace A in many applications with small loss, while being easier to compute with because it is smaller.

The matrix Chernoff bound

Theorem 28.2 (Matrix Chernoff bounds)**.** *Let $\epsilon \in (0, 1)$ and let $c > 0$ be sufficiently large. Let $\alpha = \epsilon^2 / c \log(n)$ for $c > 0$ sufficiently large. Let $X_1, \ldots, X_n$ be independent and randomized positive semidefinite matrices with $X_i \preceq \alpha I$ for all i, and $\mathbf{E}[X_1 + \cdots + X_n] = I$. Then*

$$\mathbf{P}[(1 - \epsilon)I \preceq X_1 + \cdots + X_n \preceq (1 + \epsilon)I] \geq 1 - n^{-\Omega(c)}.$$

The proofs of the matrix Chernoff bounds require technical tools beyond the scope of our course. Still let us try to offer some intuition.

Let $u \in \mathbb{R}^n$ be any unit vector. Consider the terms $\langle u, X_i u\rangle$, which (in a sense) project X_i onto the direction u. The assumptions of the matrix Chernoff bound state that

$$0 \leq \langle u, X_i u\rangle \leq \frac{\epsilon^2}{c \log n}$$

for all i, and that

$$\mathbf{E}\left[\left\langle u, \left(\sum_{i=1}^{n} X_i\right) u\right\rangle\right] = \mathbf{E}\left[\sum_{i=1}^{m} \langle u, X_i u\rangle\right] = 1.$$

If we apply the standard Chernoff bound to the independent random variables $Y_i = \langle u, X_i u\rangle$, we get

$$1 - \epsilon \leq \sum_{i=1}^{n} \langle u, X_i u\rangle \leq 1 + \epsilon$$

with probability at least $1 - n^{-\Omega(c)}$.

Thus in any single direction, the random sum $\sum_i X_i$ is well concentrated. The strength of the matrix Chernoff bound is the assertion that the random sum $\sum_i X_i$ is simultaneously well-concentrated in *all* directions. This is much stronger than can be obtained by a union bounded argument over all directions...

Sparsifying sums.

Fall 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

28.3 Exercises

Exercise 28.1. Suppose B is an $(1 \pm \epsilon)$-approximation of A. Prove that for all i, if $\mu_{A,i}$ is the ith largest eigenvalue of A, and $\mu_{B,i}$ is the ith largest eigenvalue of B, then

$$(1 - \epsilon)\mu_{B,i} \leq \mu_{A,i} \leq (1 + \epsilon)\mu_{B,i}.$$

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: http://theory.cs.princeton.edu/complexity/.

[AC09] Nir Ailon and Bernard Chazelle. "The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors". In: *SIAM J. Comput.* 39.1 (2009), pp. 302–322. URL: https://doi.org/10.1137/060673096.

[Ach01] Dimitris Achlioptas. "Database-friendly random projections". In: *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*. ACM, 2001.

[AKP+95] Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. "A Graph-Theoretic Game and Its Application to the k-Server Problem". In: *SIAM J. Comput.* 24.1 (1995), pp. 78–100. Extended abstract in DIMACS Workshop on On-Line Algorithms, 1991.

[Alo91] Noga Alon. "A Parallel Algorithmic Version of the Local Lemma". In: *Random Struct. Algorithms* 2.4 (1991), pp. 367–378. URL: https://doi.org/10.1002/rsa.3240020403. Preliminary version in FOCS, 1991.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *J. Comput. Syst. Sci.* 58.1 (1999). Preliminary version in STOC, 1996, pp. 137–147.

[And89] Arne Andersson. "Improving Partial Rebuilding by Using Simple Balance Criteria". In: *Algorithms and Data Structures, Workshop WADS '89, Ottawa, Canada, August 17-19, 1989, Proceedings*. Vol. 382. Lecture Notes in Computer Science. Springer, 1989, pp. 393–402. URL: https://doi.org/10.1007/3-540-51542-9%5C_33.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. "Expander flows, geometric embeddings and graph partitioning". In: *J. ACM* 56.2 (2009), 5:1–5:37.

[AS16] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. 4th. Wiley Publishing, 2016.

[Ban10] Nikhil Bansal. "Constructive Algorithms for Discrepancy Minimization". In: *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 2010, pp. 3–10. URL: https://doi.org/10.1109/FOCS.2010.7.

[Bar96] Yair Bartal. "Probabilistic Approximations of Metric Spaces and Its Algorithmic Applications". In: *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*. IEEE Computer Society, 1996, pp. 184–193.

[Bar98] Yair Bartal. "On Approximating Arbitrary Metrices by Tree Metrics". In: *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*. ACM, 1998, pp. 161–168.

[Bec91] József Beck. "An Algorithmic Approach to the Lovász Local Lemma. I". In: *Random Struct. Algorithms* 2.4 (1991), pp. 343–366. URL: https://doi.org/10.1002/rsa.3240020402.

[Bel97] Richard Bellman. *Introduction to Matrix Analysis (2nd Ed.)* USA: Society for Industrial and Applied Mathematics, 1997.

[BFP+72] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. "Linear Time Bounds for Median Computations". In: *Proceedings of the 4th Annual ACM Symposium on Theory of Computing, May 1-3, 1972, Denver, Colorado, USA*. ACM, 1972, pp. 119–124.

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020. Preliminary version available at https://www.cs.cornell.edu/jeh/book.pdf.

[BK15] András A. Benczúr and David R. Karger. "Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs". In: *SIAM J. Comput.* 44.2 (2015), pp. 290–319.

[BLR93a] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. "Self-Testing/Correcting with Applications to Numerical Problems". In: *J. Comput. Syst. Sci.* 47.3 (1993), pp. 549–595. Preliminary version in STOC, 1990.

[BLR93b] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. "Self-Testing/Correcting with Applications to Numerical Problems". In: *J. Comput. Syst. Sci.* 47.3 (1993), pp. 549–595. URL: `https://doi.org/10.1016/0022-0000(93)90044-W`. Preliminary version in STOC, 1990.

[Blu00] Avrim Blum. "Paging". Lecture notes from course "Approximation and Online Algorithms". Mar. 2000. URL: `http://www.cs.cmu.edu/~avrim/Approx00/lectures/lect0308`.

[BMP+98] Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. "What can you do with a Web in your Pocket?" In: *IEEE Data Eng. Bull.* 21.2 (1998), pp. 37–47. URL: `http://sites.computer.org/debull/98june/webbase.ps`.

[Bol98] Béla Bollobás. *Modern Graph Theory*. 1st ed. Graduate Texts in Mathematics 184. Springer-Verlag New York, 1998.

[Bou85] Jean Bourgain. "On Lipschitz embedding of finite metric spaces in Hilbert space". In: *Israel Journal of Mathematics* 52.1-2 (Mar. 1985), pp. 46–52.

[BP98] Sergey Brin and Lawrence Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: *Comput. Networks* 30.1-7 (1998), pp. 107–117.

[CCF02] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. "Finding Frequent Items in Data Streams". In: *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*. Vol. 2380. Lecture Notes in Computer Science. Springer, 2002, pp. 693–703. URL: `https://doi.org/10.1007/3-540-45465-9%5C_59`.

[Cha02] Moses Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. ACM, 2002, pp. 380–388.

[Che14] Chandra Chekuri. "Lecture 6 from CS 598CSC: Algorithms for Big Data". Lecture notes. 2014. URL: https://courses.engr.illinois.edu/cs598csc/fa2014/Lectures/lecture_6.pdf.

[Chu97] Fan R. K. Chung. *Spectral graph theory*. eng. Regional conference series in mathematics, no. 92. Providence, R.I: Published for the Conference Board of the mathematical sciences by the American Mathematical Society, 1997.

[CJN18] Michael B. Cohen, T. S. Jayram, and Jelani Nelson. "Simple Analyses of the Sparse Johnson-Lindenstrauss Transform". In: *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*. Vol. 61. OASICS. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 15:1–15:9. URL: https://doi.org/10.4230/OASIcs.SOSA.2018.15.

[CM05] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". In: *J. Algorithms* 55.1 (2005), pp. 58–75. URL: https://doi.org/10.1016/j.jalgor.2003.12.001. Preliminary version in LATIN 2004.

[Coo71] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. ACM, 1971, pp. 151–158.

[CS00] Artur Czumaj and Christian Scheideler. "Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász local lemma". In: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*. ACM/SIAM, 2000, pp. 30–39. URL: http://dl.acm.org/citation.cfm?id=338219.338229.

[CVM22] Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel. "Distinct Elements in Streams: An Algorithm for the (Text) Book". In: *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*. Vol. 244. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 34:1–34:6. URL: https://doi.org/10.4230/LIPIcs.ESA.2022.34.

[CW] Chris 73 and Wikipedia Commons. *WorldWideWeb Around Wikipedia*. URL: https://en.wikipedia.org/wiki/World_Wide_Web#/media/File:WorldWideWebAroundWikipedia.png.

[Din07] Irit Dinur. “The PCP theorem by gap amplification”. In: *J. ACM* 54.3 (2007), p. 12. Preliminary version in STOC, 2006.

[DKS10] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. “A sparse Johnson: Lindenstrauss transform”. In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. ACM, 2010, pp. 341–350.

[DM17] Petros Drineas and Michael W. Mahoney. “Lectures on Randomized Numerical Linear Algebra”. In: *CoRR* abs/1712.08880 (2017). arXiv: 1712.08880. URL: http://arxiv.org/abs/1712.08880.

[DS84] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*. 1st ed. Vol. 22. Mathematical Association of America, 1984. URL: https://math.dartmouth.edu/~doyle/docs/walks/walks.pdf.

[EL75] Paul Erdös and Lázló Lovász. “Problems and results on 3-chromatic hypergraphs and some related questions”. In: *Infinite and Finite Sets (to Paul Erdős on his 60th birthday)*. Ed. by A. Hajnal; R. Rado; V. T. Sós. Vol. II. North-Holland, 1975, pp. 609–627. URL: http://www.cs.elte.hu/~lovasz/scans/LocalLem.pdf.

[ER59] Paul Erdös and Alfred Rényi. “On Random Graphs I”. In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290.

[ER60] Paul Erdös and Alfred Rényi. “On the evolution of random graphs”. In: *Publ. Math. Inst. Hungary. Acad. Sci.* 5 (1960), pp. 17–61.

[Eri17] Jeff Erickson. *Hashing*. 2017. URL: https://jeffe.cs.illinois.edu/teaching/algorithms/notes/05-hashing.pdf.

[FF56] L. R. Ford and D. R. Fulkerson. “Maximal Flow Through a Network”. In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404.

[Fie73] Miroslav Fiedler. “Algebraic connectivity of graphs”. In: *Czechoslovak Mathematical Journal* 23.2 (1973), pp. 298–305.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. “Competitive Paging Algorithms”. In: *J. Algorithms* 12.4 (1991), pp. 685–699. URL: https://arxiv.org/abs/cs/0205038.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. “A tight bound on approximating arbitrary metrics by tree metrics”. In: *J. Comput. Syst. Sci.* 69.3 (2004), pp. 485–497.

[Gal14] David Galvin. *Three tutorial lectures on entropy and counting*. 2014. arXiv: 1406.7872 [math.CO]. URL: https://arxiv.org/abs/1406.7872.

[Goo89] Irving John Good. "C332. Surprise indexes and p-values". In: *Journal of Statistical Computation and Simulation* 32.1-2 (1989), pp. 90–92. eprint: https://doi.org/10.1080/00949658908811160. URL: https://doi.org/10.1080/00949658908811160.

[GR93] Igal Galperin and Ronald L. Rivest. "Scapegoat Trees". In: *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*. ACM/SIAM, 1993, pp. 165–174. URL: http://dl.acm.org/citation.cfm?id=313559.313676.

[Har11] Sariel Har-Peled. *Geometric Approximation Algorithms*. USA: American Mathematical Society, 2011.

[Har18a] Sariel Har-Peled. *Concentration of Random Variables — Chernoff's Inequality*. Lecture notes for CS574: Randomized algorithms. Apr. 2018. URL: https://sarielhp.org/teach/17/b/lec/07_chernoff.pdf.

[Har18b] Sariel Har-Peled. *On complexity, sampling, and ϵ-nets and ϵ-samples*. Lecture notes for CS574. Apr. 2018. URL: https://sarielhp.org/teach/17/b/lec/22_vc_dim.pdf.

[Har19] Sariel Har-Peled. *Quick Sort with High Probability*. Lecture notes from Fall 2019 course on randomized algorithms at UIUC. 2019. URL: https://sarielhp.org/teach/19/01_fall/lecs/l19_09_03.pdf.

[Hås01] Johan Håstad. "Some optimal inapproximability results". In: *J. ACM* 48.4 (2001), pp. 798–859. Preliminary version in STOC, 1997.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. "Expander graphs and their applications". eng. In: *Bulletin (new series) of the American Mathematical Society* 43.4 (2006), pp. 439–562.

[HNH13] Stefan Heule, Marc Nunkesser, and Alexander Hall. "HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm". In: *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*. ACM, 2013, pp. 683–692.

[HNS+95] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. "Sampling-Based Estimation of the Number of Distinct Values of an Attribute". In: *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.* Morgan Kaufmann, 1995, pp. 311–322.

[HW87] David Haussler and Emo Welzl. "epsilon-Nets and Simplex Range Queries". In: *Discret. Comput. Geom.* 2 (1987), pp. 127–151. URL: https://doi.org/10.1007/BF02187876. Preliminary version in SCG 1986.

[IM98] Piotr Indyk and Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998.* ACM, 1998, pp. 604–613. URL: https://doi.org/10.1145/276698.276876.

[JL84] William Johnson and Joram Lindenstrauss. "Extensions of Lipschitz maps into a Hilbert space". In: *Contemporary Mathematics* 26 (Jan. 1984), pp. 189–206.

[Kar90] Richard M. Karp. "The Transitive Closure of a Random Digraph". In: *Random Struct. Algorithms* 1.1 (1990), pp. 73–94.

[Kar93] David R. Karger. "Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm". In: *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA.* ACM/SIAM, 1993, pp. 21–30.

[KLL+97] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997.* ACM, 1997, pp. 654–663.

[KMM+94] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan S. Owicki. "Competitive Randomized Algorithms for Nonuniform Problems". In: *Algorithmica* 11.6 (1994), pp. 542–571. URL: https://doi.org/10.1007/BF01189993. Preliminary version in SODA, 1990.

[KN14] Daniel M. Kane and Jelani Nelson. "Sparser Johnson-Lindenstrauss Transforms". In: *J. ACM* 61.1 (2014), 4:1–4:23.

[Knu23] Donald E. Knuth. *The CVM Algorithm for Estimating Distinct Elements in Streams*. 2023. URL: https://cs.stanford.edu/~knuth/papers/cvm-note.pdf.

[Knu63] Donald Knuth. *Notes on "open" addressing*. 1963. URL: http://jeffe.cs.illinois.edu/teaching/datastructures/2011/notes/knuth-OALP.pdf.

[KS96] David R. Karger and Clifford Stein. "A New Approach to the Minimum Cut Problem". In: *J. ACM* 43.4 (1996). Preliminary version in STOC, 1993, pp. 601–640.

[Lax07] Peter D. Lax. *Linear Algebra and Its Applications*. Second. Hoboken, NJ: Wiley-Interscience, 2007.

[Lev73] Leonid A. Levin. "Universal Sequential Search Problems". In: *Problems of Information Transmission* 9.3 (1973).

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. "The Geometry of Graphs and Some of its Algorithmic Applications". In: *Combinatorica* 15.2 (1995). Preliminary version in FOCS 1994, pp. 215–245.

[LM12] Shachar Lovett and Raghu Meka. "Constructive Discrepancy Minimization by Walking on the Edges". In: *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. IEEE Computer Society, 2012, pp. 61–67. URL: https://doi.org/10.1109/FOCS.2012.23.

[LR99] Frank Thomson Leighton and Satish Rao. "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms". In: *J. ACM* 46.6 (1999), pp. 787–832.

[Men27] Karl Menger. "Zur allgemeinen Kurventheorie". In: *Fundamenta Mathematicae* 10.1 (1927), pp. 96–115.

[Mor68] Donald R. Morrison. "PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric". In: *J. ACM* 15.4 (Oct. 1968), pp. 514–534.

[Mos09] Robin A. Moser. "A constructive proof of the Lovász local lemma". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. ACM, 2009, pp. 343–350.

[MR02] Michael Molloy and Bruce Reed. *Graph coloring and the probabilistic method*. Vol. 23. Algorithms and combinatorics. Berlin Heidelberg: Springer-Verlag, 2002.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MR98] Michael Molloy and Bruce A. Reed. "Further Algorithmic Aspects of the Local Lemma". In: *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*. ACM, 1998, pp. 524–529. URL: https://doi.org/10.1145/276698.276866.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.

[MT10] Robin A. Moser and Gábor Tardos. "A constructive proof of the general Lovász local lemma". In: *J. ACM* 57.2 (2010), 11:1–11:15.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. URL: https://doi.org/10.1017/CBO9780511813603.

[Nel16] Jelani Nelson. *Load balancing, k-wise independence, chaining, linear probing*. Recorded lecture. 2016. URL: https://www.youtube.com/watch?v=WqBc0ZCU4Uw&list=PL2SOU6wwxB0uP4rJgf5ayhHWgw7akUWSf&index=3.

[Nel20] Jelani Nelson. "Sketching Algorithms". Lecture notes. Dec. 2020. URL: https://www.sketchingbigdata.org/fall20/lec/notes.pdf.

[ODo14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. URL: http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/analysis-boolean-functions.

[PBM+99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999. URL: http://ilpubs.stanford.edu:8090/422/.

[PPR09] Anna Pagh, Rasmus Pagh, and Milan Ruzic. "Linear Probing with Constant Independence". In: *SIAM J. Comput.* 39.3 (2009). Preliminary version in STOC, 2007, pp. 1107–1120.

[PT16] Mihai Patrascu and Mikkel Thorup. "On the k-Independence Required by Linear Probing and Minwise Independence". In: *ACM Trans. Algorithms* 12.1 (2016). Preliminary version in ICALP, 2010, 8:1–8:27.

[Ram30] Frank Plumpton Ramsey. "On a Problem of Formal Logic". In: *Proceedings of the London Mathematical Society* s2-30.1 (Jan. 1930), pp. 264–286.

[Rei08] Omer Reingold. "Undirected connectivity in log-space". In: *J. ACM* 55.4 (2008), 17:1–17:24. URL: https://doi.org/10.1145/1391289.1391291. Preliminary version in FOCS 2005.

[Rot16] Thomas Rothvoss. "Lecture Notes on the ARV Algorithm for Sparsest Cut". In: *CoRR* abs/1607.00854 (2016). arXiv: 1607.00854. URL: http://arxiv.org/abs/1607.00854.

[RVW00] Omer Reingold, Salil P. Vadhan, and Avi Wigderson. "Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders and Extractors". In: *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*. IEEE Computer Society, 2000, pp. 3–13. URL: https://doi.org/10.1109/SFCS.2000.892006. Preprint available at https://eccc.weizmann.ac.il/eccc-reports/2001/TR01-018/index.html.

[Sav70] Walter J. Savitch. "Relationships Between Nondeterministic and Deterministic Tape Complexities". In: *J. Comput. Syst. Sci.* 4.2 (1970), pp. 177–192. URL: https://doi.org/10.1016/S0022-0000(70)80006-X.

[Sch19] Robert Schapire. *CS511: Theoretical Machine Learning*. 2019. URL: https://www.cs.princeton.edu/courses/archive/spring19/cos511/schedule.html.

[Sha48] Claude E. Shannon. "A mathematical theory of communication". In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423.

[Spe85] Joel Spencer. “Six Standard Deviations Suffice”. eng. In: *Transactions of the American Mathematical Society* 289.2 (1985), pp. 679–706.

[Spi19] Daniel A. Spielman. “Spectral and Algebraic Graph Theory”. Draft. 2019. URL: https://www.cs.yale.edu/homes/spielman/sagt/sagt.pdf.

[Sri08] Aravind Srinivasan. “Improved algorithmic versions of the Lovász Local Lemma”. In: *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*. SIAM, 2008, pp. 611–620. URL: http://dl.acm.org/citation.cfm?id=1347082.1347150.

[SS89] Jeanette P. Schmidt and Alan Siegel. “On Aspects of Universality and Performance for Closed Hashing (Extended Abstract)”. In: *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*. ACM, 1989, pp. 355–366.

[SS90] Jeanette P. Schmidt and Alan Siegel. “The Analysis of Closed Hashing under Limited Randomness (Extended Abstract)”. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. ACM, 1990, pp. 224–234.

[ST85a] Daniel Dominic Sleator and Robert Endre Tarjan. “Amortized Efficiency of List Update and Paging Rules”. In: *Commun. ACM* 28.2 (1985), pp. 202–208. URL: https://doi.org/10.1145/2786.2793.

[ST85b] Daniel Dominic Sleator and Robert Endre Tarjan. “Self-Adjusting Binary Search Trees”. In: *J. ACM* 32.3 (1985), pp. 652–686. Preliminary version in STOC, 1983.

[Tao09] Terence Tao. *Moser’s entropy compression argument*. Aug. 2009. URL: https://terrytao.wordpress.com/2009/08/05/mosers-entropy-compression-argument/.

[Tho15a] Mikkel Thorup. “High Speed Hashing for Integers and Strings”. In: *CoRR* abs/1504.06804 (2015). arXiv: 1504.06804. URL: http://arxiv.org/abs/1504.06804.

[Tho15b] Mikkel Thorup. “Linear Probing with 5-Independent Hashing”. In: *CoRR* abs/1509.04549 (2015). arXiv: 1509.04549. URL: http://arxiv.org/abs/1509.04549.

[TM71] Myron Tribus and Edward C. McIrvine. “Energy and Information”. In: *Scientific American* 225 (1971), pp. 179–188.

[Tre16] Luca Trevisan. *Lecture Notes on Graph Partitioning, Expanders, and Spectral Methods*. Spring 2016. URL: https://lucatrevisan.github.io/books/expanders-2016.pdf.

[Vad12] Salil P. Vadhan. "Pseudorandomness". In: *Foundations and Trends in Theoretical Computer Science* 7.1-3 (2012), pp. 1–336. URL: https://people.seas.harvard.edu/~salil/pseudorandomness/pseudorandomness-published-Dec12.pdf.

[Val84a] Leslie G. Valiant. "A Theory of the Learnable". In: *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*. ACM, 1984, pp. 436–445. URL: https://doi.org/10.1145/800057.808710.

[Val84b] Leslie G. Valiant. "A Theory of the Learnable". In: *Commun. ACM* 27.11 (1984), pp. 1134–1142. URL: https://doi.org/10.1145/1968.1972.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. URL: http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities". In: *Theory of Probability & Its Applications* 16.2 (1971), pp. 264–280. eprint: https://doi.org/10.1137/1116025. URL: https://doi.org/10.1137/1116025.

[Ver18] Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018.

[Von] Vonfrisch. *Erdos generated network-p0.01.jpg*. URL: https://commons.wikimedia.org/wiki/File:Erdos_generated_network-p0.01.jpg.

[Wig09] Avi Wigderson. "Knowledge, creativity, and *P* vs *NP*". Available at https://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/AW09/AW09.pdf. 2009.

[Wika] Wikipedia. *https://en.wikipedia.org/wiki/PageRank*. Accessed Ocotober, 2020. URL: https://en.wikipedia.org/wiki/PageRank.

[Wikb] Wikipedia contributors. *Binomial distribution — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Binomial_distribution.

[Woo14a] David P. Woodruff. "Sketching as a Tool for Numerical Linear Algebra". In: *Found. Trends Theor. Comput. Sci.* 10.1-2 (2014), pp. 1–157.

[Woo14b] David P. Woodruff. “Sketching as a Tool for Numerical Linear Algebra”. In: *Found. Trends Theor. Comput. Sci.* 10.1-2 (2014), pp. 1–157. URL: https://doi.org/10.1561/0400000060.

[WS11] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011. URL: https://www.designofapproxalgs.com.

[Yua92] *18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada, Proceedings*. Morgan Kaufmann, 1992.

Appendix A

Homework assignments

AREN'T YOU SUPPOSED TO BE DOING YOUR HOMEWORK?
I'M PRETTY SURE THE ASSIGNMENT WAS OPTIONAL.
DENIAL SPRINGS ETERNAL.
IT'S NOT DENIAL. I'M JUST VERY SELECTIVE ABOUT THE REALITY I ACCEPT.
© 1992 Watterson/Distributed by Andrews McMeel Syndication
9-26 WATTERSON

Homework 0

- Due 11:59PM on Thursday, September 4.
- Your self-graded version is due exactly one week later. See page 447 for (fairly detailed) instructions.
- Solutions are posted both in Chapter B and on Piazza.
- Please be aware of Section D.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

We recommend exercises C.26–C.48 as warmup exercises, especially if probability theory is new for you.

1. Exercise C.35.

2. Exercise C.31.

3. Exercise C.19.

Homework 1

- Due 11:59PM on Thursday, September 18.
- Your self-graded version is due exactly one week later. See page 447 for (fairly detailed) instructions.
- Solutions are posted both in Chapter B and on Piazza.
- Please be aware of Section D.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

The lecture on heavy hitters raised a question about uniform sampling from streams. See exercise 2.5 to learn more.

1. Exercise 2.2

2. Exercise 2.4 (I think this one is the trickiest.)

3. Exercise 2.6

4. Exercise C.63

5. Exercise C.4

Homework 2

- Due 11:59PM on Thursday, October 2.
- Your self-graded version is due exactly one week later. See page 447 for (fairly detailed) instructions.
- Solutions are posted both in Chapter B and on Piazza.
- Please be aware of Section D.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise C.14
2. Exercise C.64
3. Exercise C.45 (I think this one is the trickiest.)
4. Exercise C.66
5. Exercise C.9
6. Optional challenge (just for fun): Exercise C.17

Homework 3

- Due 11:59PM on Thursday, October 16.
- Your self-graded version is due exactly two weeks later (for this homework only, on account of the midterm). See page 447 for (fairly detailed) instructions.
- Solutions are posted both in Chapter B and on Piazza.
- Please be aware of Section D.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise C.12
2. Exercise C.8
3. Exercise C.20
4. Exercise C.21
5. Exercise C.57

Homework 4

- *These problems are only tentative until this disclaimer is removed.*
- Due 11:59PM on Thursday, November 6.
- Your self-graded version is due exactly one week later. See page 447 for (fairly detailed) instructions.
- Solutions are posted both in Chapter B and on Piazza.
- Please be aware of Section D.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise C.53
2. Exercise C.65
3. Exercise C.41
4. Exercise C.30

Homework 5

- *These problems are only tentative until this disclaimer is removed.*
- Due 11:59PM on Thursday, November 20.
- Your self-graded version is due exactly one week later. See page 447 for (fairly detailed) instructions.
- Solutions are posted both in Chapter B and on Piazza.
- Please be aware of Section D.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 16.1
2. Exercise 17.1
3. Exercise 18.2
4. Exercise 19.2

Homework 6

- *These problems are only tentative until this disclaimer is removed.*
- Due 11:59PM on Thursday, December 4.
- Your self-graded version is due exactly one week later. See page 447 for (fairly detailed) instructions.
- Solutions are posted both in Chapter B and on Piazza.
- Please be aware of Section D.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 20.2
2. Exercise 21.7
3. Exercise 22.1
4. Exercise 23.6
5. Problem 4.8 in [Vad12]. (I think this one is the trickiest.)

Appendix B

Selected solutions

B.1 Homework 0

Exercise C.35. Suppose you repeatedly flip a coin that is heads with fixed probability $p \in (0, 1)$.

1. What is the expected number of coin flips until you obtain one heads?[1] Prove your answer.[2]

2. What is the expected number of coin flips until you obtain two heads? Prove your answer.

3. For general $k \in \mathbb{N}$, what is the expected number of coin tosses until you obtain k heads? Prove your answer.

Solution to exercise C.35.

1. Let $X \in \mathbb{N}$ be the random variable reflecting the number of flips until heads.

   Observe that, if the first coin toss is tails, then we are again flipping coins until we get heads. Consequently

   $$\mathbf{E}[X \mid \text{first coin is tails}] = 1 + \mathbf{E}[X].$$

   Therefore

   $$\begin{aligned}\mathbf{E}[X] &= p\,\mathbf{E}[X \mid \text{first coin is heads}] + (1-p)\,\mathbf{E}[X \mid \text{first coin is tails}] \\ &= p + (1-p)(1 + \mathbf{E}[X]) \\ &= 1 + (1-p)\,\mathbf{E}[X].\end{aligned}$$

[1]If the first toss is heads, that counts as one coin flip. If the first toss is tails and the second toss is heads, that counts as two coin tosses. Etc. It may be helpful to first think about a fair coin, where $p = 1/2$.

[2]*Hint:* There is an easy way and a hard way to solve this problem.

This is solved by

$$\mathbf{E}[X] = \frac{1}{p}.$$

Painful approach. The explicit expected value formula gives

$$\mathbf{E}[X] = \sum_{i=1}^{\infty} i\,\mathbf{P}[X = i] = \sum_{i=1}^{\infty} ip(1-p)^{i-1}.$$

To evaluate this sum (carefully), we have

$$\begin{aligned}
\sum_{i=1}^{\infty} ip(1-p)^{i-1} &= \lim_{n\to\infty} p \sum_{i=1}^{n} \frac{d}{dp} - (1-p)^i \\
&= \lim_{n\to\infty} -p\frac{d}{dp}\sum_{i=1}^{n}(1-p)^i \\
&= \lim_{n\to\infty} -p\frac{d}{dp}\frac{1-(1-p)^{n+1}}{p} \\
&= \lim_{n\to\infty} -p\frac{n(1-p)^n p - (1-(1-p)^{n+1})}{p^2} \\
&= \frac{1}{p}.
\end{aligned}$$

2. (See part 3 below.)

3. Let X_i be the number of flips between the $(i-1)$th heads and the ith heads. Then $X_1 + \cdots + X_k$ is the total number of flips until k heads. We have

$$\mathbf{E}[X_1 + \cdots + X_k] = \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_k] = \frac{k}{p}.$$

Rubric. For this and other problems, there may be other approaches to the problems that aren't covered by the rubric. In general, if the solution is correct and adequately proven, it gets full credit. For partial credit, we try our best to be fair in a way that is consistent with the breakdowns below.

5 pts. Part 1.

- Nice approach.
  - **1 pt.** Observing that after first tails, the rest repeats the same experiment.
  - **1 pt.** Break down $\mathbf{E}[X]$ w/ conditional probabilities.
  - **1 pt.** Solve recursive equation
  - **2 pts.** Get $1/p$.
- Painful approach.
  - **1.5 pts.** Explicitly modeling the expected value.
  - **.5 pts.** Either justify interchanging the derivative with the infinite sum, or explicitly state the limit and interchagne with the inner finite sum.
  - **1 pt.** Apply geometric sum formula.
  - **2 pts.** Get $1/p$.

3 pts. Part 2.
- **1 pt.** Modeling with one X_i for each heads.
- **1 pt.** Applying linearity of expectation.
- **1 pt.** Getting $2/p$.

2 pts. Part 3.
- **2/3 pts.** Modeling with one X_i for each heads.
- **2/3 pts.** Applying linearity of expectation.
- **2/3 pts.** Getting k/p.

Exercise C.31. Please see the Piazza post for the solution and rubric to Exercise C.31.

Exercise C.19. This exercise is about a simple randomized algorithm for *verifying* matrix multiplication. Suppose we have three $n \times n$ matrices A, B, C. We want to verify if $AB = C$. Of course one could compute the product AB and compare it entrywise to C. But multiplying matrices is slow: the straightforward approach takes $O(n^3)$ time and there are (more theoretical) algorithms with running time roughly $O(n^{2.37\cdots})$. We want to test if $AB = C$ in closer to n^2 time.

The algorithm we analyze is very simple. Select a point $x \in \{0,1\}^n$ uniformly at random. (That is, each $x_i \in \{0,1\}$ is an independently sampled bit.) Compute $A(Bx)$ and Cx, and compare their entries. (Note that it is much faster to compute $A(Bx)$ than AB.) If they are unequal, then certainly $AB \neq C$ and we output `false`. Otherwise we output `true`. Note that the algorithm is always correct if $AB = C$, but could be wrong when $AB \neq C$. We will show that if $AB \neq C$, the algorithm is correct with probability at last $1/2$.

1. Let $y \in \mathbb{R}^n$ be a fixed nonzero vector, and let $x \in \{0,1\}^n$ be drawn uniformly at random. Show that $\langle x, y\rangle \overset{\text{def}}{=} \sum_{i=1}^n x_i y_i \neq 0$ with probability at least $1/2$. [3]

2. Use the preceding result to show that if $AB \neq C$, then with probability at least $1/2$, $ABx \neq Cx$.[4]

3. Suppose we want to decrease our probability of error to (say) $1/n^2$. Based on the algorithm above, design and analyze a fast randomized algorithm with the following guarantees.

   - If $AB = C$, then it always reports that $AB = C$.
   - If $AB \neq C$, then with probability of error at most $1/n^2$, it reports that $AB \neq C$.

   (Your analysis should include the running time as well.)

Solution to exercise C.19.

1. Let y_k be the last nonzero coefficient of y. Let $\alpha = \sum_{i=1}^{k-1} x_i y_i$ be the partial sum over the first $k-1$ coordinates, and consider the probability of $\langle x, y\rangle = 0$ conditional on α. If $\alpha = 0$, then $\langle x, y\rangle \neq 0$ if $x_k = 1$. If $\alpha \neq 0$, then $\langle x, y\rangle \neq 0$ if $x_k = 0$. Thus

$$\begin{aligned}\mathbf{P}[\langle x,y\rangle \neq 0] &= \mathbf{P}[\langle x,y\rangle \neq 0 \,|\, \alpha = 0]\,\mathbf{P}[\alpha = 0] + \mathbf{P}[\langle x,y\rangle \neq 0 \,|\, \alpha = 1]\,\mathbf{P}[\alpha = 1] \\ &\geq \mathbf{P}[x_k = 1]\,\mathbf{P}[\alpha = 0] + \mathbf{P}[x_k = 0]\,\mathbf{P}[\alpha = 1] \\ r &\geq \frac{1}{2}\mathbf{P}[\alpha = 0] + \frac{1}{2}\mathbf{P}[\alpha = 1] = \frac{1}{2},\end{aligned}$$

   as desired.

2. If $AB \neq C$, then there is some row i such that the ith row of AB does not equal the ith row of C. Recall that $(ABx)_i$ and $(Cx)_i$ are the inner product of x with the ith row of AB and C, respectively. By the previous part, with probability at least $1/2$, $(ABx)_i \neq (Cx)_i$ (in which case $ABx \neq Cx$).

[3]*Hint: Suppose for simplicity that the last coordinate of y is nonzero. It might help to imagine sampling the first $n-1$ bits and computing the partial sum $S_{n-1} = \sum_{i=1}^{n-1} x_i y_i$ first, before sampling x_n and adding $x_n y_n$. Formally your analysis may involve some conditional probabilities. (And what about the case where $y_n = 0$?)*

[4]Even if you haven't solved part 1 you may assume it to be true.

3. We repeat the algorithm from part 2 $O(\log n)$ times, answering that $AB = C$ iff $ABx = Cx$ for each randomly sampled x. If $AB = C$, then $ABx = Cx$ for all x and we correctly respond that $AB = C$. If $AB \neq C$, then for each x, we have $ABx = Cx$ with probability at most 1/2. The probability that $ABx = Cx$ for all n trials is therefore $1/\operatorname{poly}(n)$. The total running time is $O(n^2)$, since it takes $O(n^2)$ time to compute ABx and Cx, and we rerun the algorithm $O(\log n)$ times.

Rubric.

4 pts. Part 1.
- **1 pts.** Generalizing from the special case where the last coefficient is nonzero by focus on the last nonzero coefficient.
- **1.5 pts.** Analysis conditioning on when the previous sum is zero.
- **1.5 pts.** Analysis conditioning on previous sum is nonzero.

3 pts. Part 2.
- **1.5 pts.** Recognizing that if $A \neq BC$, at least one row of the difference is a nonzero vector.
- **1.5 pts.** Correctly applying part 1 to this row.

3 pts. Part 3.
- **1 pt.** Repeating the algorithm $O(\log n)$ times.
- **1 pt.** Accept iff all independent trials accept.
- **.5 pts.** Show that probability of error becomes $1/\operatorname{poly}(n)$.
- **.5 pts.** Analyze the running time.

B.2 Homework 1

Exercise 2.2. Show that the construction given in Section 2.2 is indeed a universal hash function, using the steps listed below.

To recall the construction, we randomly construct a function $h : [n] \to [k]$ as follows. First, let p be any prime number $> n$. Draw $a \in \{1, \dots, p-1\}$ uniformly at random, and draw $b \in \{0, \dots, p-1\}$ uniformly at random. We define a function $h(x)$ by

$$h(x) = ((ax + b) \mod p) \mod k.$$

1. Let $x_1, x_2 \in [n]$ with $x_1 \neq x_2$, and let $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$. Show that the system of equations

$$\begin{aligned} ax_1 + b = c_1 \mod p \\ ax_2 + b = c_2 \mod p \end{aligned}$$

uniquely determines $a \in \{1, \dots, p-1\}$ and $b \in \{0, \dots, p-1\}$.[5]

- Step 1 implies that the map $(a, b) \mapsto (ax_1 + b \mod p, ax_2 + b \mod p)$ is a bijection between $\{1, \dots, p-1\} \times \{0, \dots, p-1\}$ and $\{(c_1, c_2) \in \{0, \dots, p-1\} : c_1 \neq c_2\}$.

2. Let $x_1, x_2 \in [n]$ with $x_1 \neq x_2$, and let $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$. Show that

$$\mathbf{P}[ax_1 + b = c_1 \mod p,\, ax_2 + b = c_2 \mod p] = \frac{1}{p(p-1)}.$$

(Here the randomness is over the uniformly random choices of a and b.)

3. Fix $x_1, x_2 \in [n]$ with $x_1 \neq x_2$, and $c_1 \in \{0, \dots, p-1\}$. Show that

$$\sum_{\substack{c_2 \in \{1,\dots,p\} \\ c_2 \neq c_1 \mod p \\ c_1 = c_2 \mod k}} \mathbf{P}[ax_1 + b = c_1 \mod p,\, ax_2 + b = c_2 \mod p] \leq \frac{1}{pk}.$$

- The LHS represents $\mathbf{P}[ax_1 + b = c_1 \mod p \text{ and } h(x_2) = h(x_1)]$.[6][7]

4. Finally, show that $\mathbf{P}[h(x_1) = h(x_2)] \leq \frac{1}{k}$.

Solution to exercise 2.2.

[5]Here it is helpful to know that division is well-defined on the set of integers modullo p when p is prime. More precisely, "a/b" is defined as the unique integer c such that $bc = a$.

[6]Here we note that for $x_1 \neq x_2$, $ax_1 + b = ax_2 + b_2 \mod p$ iff $a = 0$.

[7]*Hint:* You may want to show that the number of values $c_2 \in [p]$ such that $c_1 = c_2 \mod k$ is $\leq \frac{p-1}{n}$.

1. Subtracting one equation from the other, we have

$$a(x_1 - x_2) = c_1 - c_2 \mod p.$$

Dividing by $x_1 - x_2$ (which is nonzero) we have

$$a = \frac{c_1 - c_2}{x_1 - x_2}.$$

Now b is determined as $b = c_1 - ax_1$.

2. By part 1, there is a unique choice of $a \in \{1, \ldots, p-1\}$ and $b \in \{0, \ldots, p-1\}$ such that $ax_1 + b = c_1 \mod x$ and $ax_2 + b = c_2 \mod p$. The probability of sampling these particular values of a and b is $1/p(p-1)$.

3. We first observe that the number of values $c_2 \in [p]$ such that $c_1 = c_2 \mod k$ is at most $(p-1)/k$. Indeed, for each interval $[(i-1)k + 1, ik]$ there is exactly one such value (namely, $(i-1)p + (c_1 \mod k)$), and $\lceil p/k \rceil$ such intervals cover the range $[p]$. One of those values is c_1, which leaves at most $\lceil p/k \rceil - 1 \leq (p-1)/k$ values $c_2 \in [p]$ such that $c_1 \neq c_2$ and $c_2 = c_1 \mod k$.

Now, we have

$$\sum_{\substack{c_2 \in \{1,\ldots,p\} \\ c_2 \neq c_1 \\ c_1 = c_2 \mod k}} \mathbf{P}[ax_1 + b = c_1 \mod p,\, ax_2 + b = c_2 \mod p] \leq \frac{p-1}{k} \cdot \frac{1}{p(p-1)} = \frac{1}{pk}.$$

4. We have

$$\mathbf{P}[h(x_1) = h(x_2)] = \sum_{c_1 \in [p]} \mathbf{P}[ax_1 + b = c_1 \text{ and } h(x_1) = h(x_2)] \overset{(a)}{\leq} \sum_{c_1 \in [p]} \frac{1}{pk} = \frac{1}{k}.$$

Here (a) is by part 3.

Rubric.
3 pts. Part 1
- **1 pt.** Getting the right value for a.
- **1 pt.** Getting the right value for b.
- **1 pt.** Showing the work/algebra to get the right values.

2 pts. Part 2
- **1 pt.** Observing that there's a unique choice of a and b.
- **1 pt.** Calculate the right probability.

4 pts. Part 3
- **2 pts.** Proving the hint.
- **2 pts.** Using the hint correctly.

1 pt. Part 4
- **.5 pts.** Union bound.
- **.5 pts.** Apply part 3.

Exercise 2.4. In this exercise, we develop a refined analysis that can reduce the additive error substantially in (arguably realistic) settings when the stream is dominated by heavy hitters.

Let S denote the sum of frequency counts of all elements that are *not* ϵ-heavy hitters:

$$S = \sum_{e:p_e<\epsilon} f_e.$$

Note that $S \leq m$, and S might be much less than m when the stream is dominated by heavy hitters.

Show that, by increasing the parameter w (in the `count-min` data structure) by a constant factor, and still using universal hash functions, one can estimate the frequency of every element with additive error at most ϵS with high probability in $O(\log(n)/\epsilon)$ space.[8]

Solution to exercise 2.4. The high level idea is as follows. Fix an element e. Consider a single instance of the `hashed-counters` data structure with $w = \lceil 4/\epsilon \rceil$. We have additive error ϵS if the following holds:

1. e does not collide with any of the ϵ-heavy hitters.
2. The total noise from the remaining non-heavy hitters is at most ϵS.

We will show that each of the items above occur with probability at least 3/4. If so, then by the union bound (with respect to the negated events), both occurs with probability 1/2. That is, with probability at least 1/2, we have additive error at most ϵS. We amplify to $1/\operatorname{poly}(n)$ probability of error by making $O(\log(n))$ copies.

We first analyze the probability that e does not collide with any of the heavy hitters. Let $D \subseteq [n]$ denote the set of heavy-hitters, we have $|D| \leq 1/\epsilon$. For a single heavy hitter $d \in D$, the probability that d collides with e is

$$\mathbf{P}[h(d) = h(e)] \leq \frac{1}{w}$$

since h is universal. Taking the union bound over all $\ell \leq 1/\epsilon$ heavy hitters,

$$\mathbf{P}[h(e) = h(d) \text{ for some } d \in D] \leq \frac{\ell}{w} \leq \frac{1}{4}.$$

Now we analyze the additive error from all non-heavy hitters. Let X denote the value of $A[h(e)]$ excluding the contributions from the heavy hitters. We want to show that $X - f_e \leq \epsilon S$ with probability at least 3/4. By (b) linearity of expectation and (c) universality of h, we have

$$\mathbf{E}[X - f_e] \overset{\text{(b)}}{=} \sum_{\substack{c \neq e \\ c \notin D}} f_d \, \mathbf{P}[h(d) = h(e)] \overset{\text{(c)}}{\leq} \frac{1}{w} \sum_{\substack{c \neq e \\ c \notin D}} f_d = \frac{S}{w} \leq \frac{\epsilon S}{4}.$$

[8]*Hint*: It might be helpful to think about the special case of $S = 0$.

By Markov's inequality,

$$\mathbf{P}[X - f_e \geq \epsilon S] \leq \mathbf{P}[X - f_e \geq 4\,\mathbf{E}[X - f_e]] \leq 1/4,$$

as desired.

Remark B.1. We analyze two events: not colliding with other ϵ-heavy hitters, and based on noise from non-heavy hitters. Above we analyzed them separately and them combined them via the union bound.

Since we are using limited randomness, you should be careful about how you are combining the two events. If you are using universal hash functions, then you cannot analyze one event conditional on the other – the other event "uses up" the randomness guarantee.

Rubric.

4 pts. Proving the element doesn't collide with any heavy hitter with probability (say) 1/4.
- **1 pt.** Observe that w is a constant factor bigger than the number of heavy hitters.
- **2 pts.** Analyze probability of colliding with a single heavy hitter (based on universality of hash function).
- **1 pt.** Union bound.

4 pts. Proving that the error from non-heavy hitters is ϵS with probability (say) 1/4.
- **3 pts.** Analyzing expected error from non-heavy hitters.
  - **1 pt.** Linearity of expectation.
  - **1 pt.** Universality of hash function.
  - **1 pt.** Getting the right bound. (Some constant factor less than ϵS).
- **1 pt.** Use Markov's inequality to bound the probability of large error.

1 pt. Combine the two events above with the union bound.

1 pt. Amplification.

Exercise 2.6. This exercise gives a different and interesting application of hashing to string matching. In string matching, we have a long text string $T[1..n]$, and a smaller search string $S[1..k]$, and we want to decide if S occurs in T. For simplicity we assume these are bit strings, but it is easy to generalize to larger alphabets.

The naive approach directly compares $S[1..k]$ to each length-k substring $T[i, ..., i+k-1]$, and takes $O(nk)$ time. A more sophisticated algorithm due to Knuth, Morris, and Pratt compiles S into a deterministice finite automaton A_S of size $O(k)$, and uses the automaton A_S to search T in $O(n)$ time. Here we take a different approach based on randomization.

Now, a k-bit string $x \in \{0,1\}^k$ can be thought of as an integer

$$2^{k-1}x_1 + 2^{k-2}x_2 + \cdots + 2x_{k-1} + x_k$$

between 0 and $2^k - 1$. One might try to compute all the integers for the k-bit substrings $T[1..k], T[2,,k+1], ..., T[n-k+1..n]$ of T and compare each to the integer form of S. But this is really no different than the naive approach of direct comparison, since the integers are k-bits long.

Suppose instead we took these integers modulo a prime p drawn from a range $\{2,\dots,q\}$, for sufficiently large q. Consider the hash function $h : \{0,1\}^k \to \mathbb{Z}_{\geq 0}$ defined by

$$h(x_{1..k}) = 2^{k-1}x_1 + 2^{k-2}x_2 + \cdots + 2x_{k-1} + x_k \mod p$$

for a randomly selected prime number p. h is a *rolling* hash function: as we "shift" the hash function by 1-bit from (say) bits $1, ..., k$ to bits $2, ..., k+1$, we need only update

$$h(x_{2..k+1}) = 2(h(x_{1..k}) - 2^{k-1}x_1) + x_{k+1} \mod p$$

with a constant number of arithmetic operations, instead of $O(k)$ operations to compute it from scratch.

The goal is to use the rolling hash function to design and analyze a fast randomized algorithm for string matching. The problem of course is collisions between distinct substrings, and the probability of collision depends on the random selection of p. The following facts about prime numbers may be helpful:

- By the prime factorization theorem, every integer can be represented uniquely as a product of prime numbers.
- By the prime number theorem, there are $(1 - o(1))n/\ln n$ primes between 1 and n.
- There is a deterministic polynomial time algorithm for verifying if a number is prime, and a faster randomized algorithm that succeeds with high probability. In particular, there is a deterministic algorithm that runs in $\tilde{O}(k^6)$ time for k-bit integers.

1. Let $x, y \in \{0,1\}^k$ be two distinct k-bit strings interpreted as integers between 0 and $2^k - 1$. Observe that $x = y \mod p$ iff p divides $|x - y|$, which is again an integer between 0 and $2^k - 1$. Prove that there are at most $\log_2(|x - y|)$ distinct prime numbers dividing $|x - y|$.
2. Suppose p is a random prime number from the range $\{2,\dots,q\}$ for some value q. How large does q need to be to guarantee that p does not divide $|x - y|$ with probability (say) at least $1/2$?

3. Using the observations from the previous 2 problems, design and analyze a randomized algorithm searching for S in T that runs in $O(n \operatorname{poly}(\log k))$ time (the faster the better). (Your algorithm should always be correct, and take $O(n \operatorname{poly}(\log k))$ time in expectation. You can assume that arithmetic operations modulo p takes $O(\log p)$ time.)[9]

Solution to exercise 2.6.

Part 1. Let $|x-y| = p_1 \cdots p_\ell$ be the unique prime decomposition of $|x-y|$. Since each prime is at least 2,

$$\log_2(x-y) = \sum_{i=1}^{\ell} \log_2(p_i) \geq \ell.$$

Part 2. Let $q = Ck \log k$ for a sufficiently large constant C. Then there are at least $(1-o(1))(q/\log q) \geq 2\log(k)$ primes in $\{1, ..., k\}$. Since $|x-y| < 2^k$, at most $\log(k)$ of these primes divide $|x-y|$. All put together, the probability that a uniformly random prime between 1 and q divides $|x-y|$ is at most $1/2$.

Part 3. We give two variations of essentially the same algorithm. The high level approach is to construct a rolling hash function with probability of error at most $1/k$. Now process the length-k windows of T with the rolling hash, updating the hash in $\operatorname{poly}(\log k)$ time per window. If a k-window hash matches the hash for S, then directly compare the window with S to see if they are equal. If they are actually equal, then we are done. If they are not equal, then we continue to the next window in T. False positives occur with probability with $1/k$, and comparing exactly takes $O(k)$ time, so we spend $O(n)$ time in expectation double-checking false positives. Observe that the double-checking step ensures that the overall algorithm is always correct.

The first algorithm generates a rolling hash by creating $O(\log k)$ individual rolling hash functions, each of which use a random prime between 1 and $O(k \log k)$, and has probability of error $1/2$. Jointly they have error probability at most $1/k$, and take $O\left(\log^2 k\right)$ time to update.

[9] *Hint:* You might first try to get a $O\left(n \log n \log^{O(1)} k\right)$ time algorithm that succeeds with high probability. (This will still get most of the points.) Getting it down to $O\left(n \log^{O(1)} k\right)$ time is a little trickier.

```
Rabin-Karp-1($T[1..n]$, $S[1..k]$)
1. For $i = 1, \dots, \ell$ where $\ell = O(\log k)$:
   A. $p_i \leftarrow$ random-prime(q) for $q = O(k \log k)$.
   B. define the rolling hash function $h_i(x) = x \mod p_i$.
2. Let $h(x) = (h_1(x), \dots, h_\ell(x))$.
3. For $j = 1, ..., n - k + 1$:
   A. If $h(T[j..k + j - 1]) = h(S)$
      // $O\left(\log^2 k\right)$ time to update $H(t[j..k + j - 1])$
      1. If $T[j..k + j - 1] = S$, then return $(j, j + k - 1)$        // $O(k)$ time
4. Return false
```

Here `random-prime` selects a random prime by uniformly sampling a number between 2 and q and testing if it is a prime. Each random number is a prime with probability $\Omega(1/\log(q))$, so it takes $O(\log q)$ iterations on average to find a prime. Since testing a prime takes $O\left(\log^6 q\right)$ time, `random-prime` takes $O\left(\log^7 q\right)$ time in expectation.

```
RandomPrime($q$)
1. Repeatedly:
   A. Draw $p \in \{2, \dots, q\}$ uniformly at random.
   B. If $p$ is prime then return $p$.        // $O\left(\log^6 q\right)$ time to test if $p$ is prime.
```

Altogether, this first algorithm takes $O\left(n \log^2 k + \log^7 k\right)$ time in expectation.

The second approach generates a rolling hash with a single random prime between 2 and $O(k^2 \log k)$. This hash function has error probability at most $1/k$, and takes $O(\log k)$ time to process and update.

```
Rabin-Karp-2($T[1..n]$, $S[1..k]$)
1. $p \leftarrow$ random-prime(q) for $q = O(k^2)$.
2. define the rolling hash function $h(x) = x \mod p$.
3. For $j = 1, ..., n - k + 1$:
   A. If $h(T[j..k + j - 1]) = h(S)$
      // $O(\log k)$ time to update $H(t[j..k + j - 1])$
      1. If $T[j..k + j - 1] = S$, then return $(j, j + k - 1)$        // $O(k)$ time
4. Return false
```

This second algorithm takes $O\left(n \log k + \log^7 k\right)$ time in expectation.

Rubric.

1 pt. Part 1

- Basic reason: Each prime factor contributes at least 2 multiplicatively.

2 pts. Part 2

- Apply the prime number theorem and choose q large enough that $q/\log q \geq 2\log|x-y|$.

7 pts. Part 3

3 pts. Describing the algorithm, with the following parts.

- Constructing a hash algorithm with probability of error $(1/k)$ or $(1/n)$ by either:
  - Describing how to construct a single-hash function with $q = O(k\log k)$ get probability of error 1/2, and then amplifying by repetition to reduce the error probability to $1/k$ or $1/\operatorname{poly}(n)$; or
  - Choosing q large enough $(\Omega(k^2\log k))$ so that the probability of error is at most $1/k$ or $1/\operatorname{poly}(n)$.

  Either way, the solution should also explain and analyze how to generate a random prime.
- Scanning T with the rolling hash.
- Double checking for false positives.

Correctness analysis (of whatever algorithm you describe).

- Analyzing the error probability of your hash (either amplifying constant error, or directly getting small error with a larger prime.)
- Correcting from double checking false positives.

1 pt. Running time analysis (of whatever algorithm you describe).

- For an algorithm that uses double checking, show how getting the probability of error down to $1/k$ pays for the $O(k)$ time spent double-checking each false positive.

Exercise C.63. The goal of this exercise is to show how to get constant time access for n keys with $O(n)$ space, using only universal hash functions. We will require the following fact that we ask you to prove.

1. Let $h : [n] \to [k]$ be a *universal* hash function. Show that for $k \geq n^2$, h has no collisions with probability $\geq 1/2$.

Now we describe the data structure. We first allocate an array $A[1..n]$ of size n. We have one universal hash function h_0 into $[n]$. If we have a set of (say) k collisions at an array cell $A[i]$, rather than making a linked list of length k, and we build another hash table, with a new universal hash function h_i, of size k^2, with no collisions (per part 1). (We may have to retry if there is a collision.) If the total size (summing the lengths of the first arrray and each of the second arrays) comes out to bigger than (say) $5n$, we try again.

2. For each $i = 1, \ldots, n$, let k_i be the number of keys that hash to the ith cell. We have

$$\text{(sum of array sizes of our data structure)} \leq n + \sum_{i=1}^{n} k_i^2.$$

Show that[10]

$$\sum_{i=1}^{n} k_i^2 \leq n + O(\text{total \# of collisions (w/r/t } h_0\text{)}).$$

3. Show that

$$\mathbf{E}[\text{total \# of collisions (w/r/t } h_0\text{)}] \leq n/2.$$

4. Show that

$$\mathbf{P}[\text{(sum of all array sizes)} > Cn] \leq 1/2$$

for some constant $C > 0$. ($C = 5$ is possible.)

Taken together, steps 1 to 3 above show that this approach will build a "perfect" hash table over the n keys in $O(n)$ space with probability of success at least $1/2$, using only universal hash functions. Even if it fails to work, we can then keep repeating the construction until it succeeds. This approach works better in *static settings*, when the set of keys is fixed.

Solution to exercise C.63.

Part 1. Any two items collides with probability at most $1/k \leq 1/n^2$. Taking the union bound over all $\binom{n}{2}$ pairs of items, the probability of having at least one collision is at most

$$\binom{n}{2} \frac{1}{n^2} = \frac{n(n-1)}{2n^2} \leq \frac{1}{2}.$$

Thus the probability of having no collisions is at most $1/2$.

[10] Here a "collision" is an unordered pair of keys with the same hash. The $O(\cdots)$ means you can choose whatever constant you find convenient; 2 is possible.

Part 2. Suppose there are k_i elements in slot i. If $k_i \leq 1$, then there are no collisions. Otherwise, there are $\binom{k_i}{2} \geq (k_i^2 - 1)/2$ collisions among these elements. Thus

$$\sum_i k_i^2 \leq n + \sum_{i:k_i>1} \left(k_i^2 - 1\right) \leq n + \sum_{i:k_i>1} 2\binom{k_i}{2} = n + 2(\text{total \# of collisons}).$$

Part 3. Any two items collides with probability at most $1/n$. By linearity of expectation over all $\binom{n}{2}$ pairs of items, the expected number of collisions is

$$\binom{n}{2}\frac{1}{n} = \frac{n(n-1)}{2n} \leq \frac{n}{2}.$$

Part 4. We have

$$\mathbf{E}[\text{total size}] \leq n + \sum_i \mathbf{E}\left[k_i^2\right] \leq 2n + 2\,\mathbf{E}[\text{total \# of collisions with respect to } h_0] \leq 3n.$$

By Markov's inequality,

$$\mathbf{P}[\text{total size} > 6n] \leq 1/2,$$

as desired.

Rubric.
2 pts. Part 1
  1 pt. Single collision probability (invoking universality)
  1 pt. Union bound
3 pts. Part 2

- Important to distinguish $k_i = 1$ and $k_i > 1$.
- Needs to relate collisions at each slot to the number of collisions.

2 pts. Part 3
  1 pt. Analyze single item's collision.
  1 pt. Linearity of expectation of all pairs.
3 pts. Part 4
  2 pts. Compute $\mathbf{E}$[total size].
  1 pt. Apply Markov's inequality.

Exercise C.4. Let $G = (V, E)$ be an undirected graph. For $k \in \mathbb{N}$ a *k-cut* is a set of edges whose removal disconnects the graph into at least k connected components. Note that for

$k \geq 3$, the minimum k-cut problem cannot easily be reduced to (s,t)-flow. In fact, the problem is NP-Hard when k is part of the input.[11]

1. Briefly describe how to modify the `random-contractions` to return a k-cut.[12]
2. Analyze the probability that your modified algorithm returns a minimum k-cut.[13]
3. Describe and analyze an algorithm, using your modified `random-contractions` as a subroutine, that computes a minimum k-cut with high probability in $O\left(n^{c_1 k} \log^{c_2} n\right)$ time for constants c_1 and c_2. (We leave it to you to identify these constants; as usual, the faster the running time, the better.)
4. How does your algorithm relate to the preceding statement that k-cut is NP-Hard when k is part of the input?

Solution to exercise C.4.

Part 1. We run the contraction algorithm until there are $\ell = 2k-1$ vertices remaining, and then try all possible k-cuts induced by this graph. Note that there are at most k^{2k-1} possible k-cuts induced by $2k-1$ vertices, and it takes $O(k^2)$ time to compute the value of each one (as there are at most $O(k^2)$ edges remaining).

Part 2. Fix a minimum k-cut. We first observe that

$$c(E) \geq \frac{2n}{k-1}\lambda$$

Indeed, every $k-1$ vertices $v_1, \dots, v_{k-1} \in V$ has total weighted degree $\geq \lambda$ since $\delta(v_1) \cup \delta(v_2) \cup \cdots \cup \delta(v_{k-1})$ is a cut. Moreover, an edge $e = \{u,v\}$ is cut by such a partition only if at least one if its two endpoints are in the set $\{v_1, \dots, v_{k-1}\}$. There are at most $2\binom{n-1}{k-2}$ such cases for fixed e, since after choosing one of the two endpoints, we still have to $k-2$ vertices from $n-1$ remaining. Thus

$$\binom{n}{k-1}\lambda \leq \sum_{\substack{v_1,\dots,v_{k-1} \\ \text{distinct}}} c(\delta(v)) \leq 2\binom{n-1}{k-2} c(E).$$

Now we have

$$\frac{\lambda}{c(E)} \leq 2\frac{(n-1)!}{(k-2)!(n-k+1)!}\frac{(k-1)!(n-k+1)!}{n!} = 2\frac{k-1}{n}$$

[11]You might find it helpful to focus on the case $k=3$ and then generalize afterwards. Although we ask you to work out the dependency on k, conceptually, it might help to think of k as being relatively small compared to n.

[12]Your algorithm design may be informed by your calculations in part 2.

[13]You may want to pattern your analysis after the one for minimum (2-)cut; in particular, you may want to develop analogs for Lemmas 4.2 and 4.3.

Thus, for a randomly drawn edge e, we have

$$\mathbf{P}[e \in C^\star] \leq \frac{2(k-1)}{n}.$$

For $k \in \mathbb{Z}_{\geq 0}$, let E_i be the event that we have not sampled $C^\star$ after i iterations. We have

$$\mathbf{P}[E_{i+1} \,|\, E_i] \geq 1 - \frac{2(k-1)}{n-i} = \frac{n - 2(k-1) - i}{n-i}$$

Let $\ell = (2k-1)$ be a parameter TBD.

$$\mathbf{P}[E_{n-\ell}] = \prod_{i=1}^{n-\ell} \mathbf{P}[E_i \,|\, E_{i-1}] \geq \prod_{i=0}^{n-\ell-1} \frac{n-2(k-1)-i}{n-i} = \frac{(n-(2k-1))!}{(\ell - 2(k-1))!} \frac{\ell!}{n!}$$

In particular, for $\ell = (2k-1)$, we have

$$\mathbf{P}[E_{n-\ell}] \geq \frac{(n-(2k-1))!(2k-1)!}{n!} = \frac{1}{\binom{n}{(2k-1)}}.$$

Thus the algorithm succeeds with probability $1/\binom{n}{(2k-1)}$.

Part 3. We repeat the algorithm above $O\Big(\binom{n}{(2k-1)} \log(n)\Big) \leq O\Big(n^{2k-2} \log(n)\Big)$ times. The probability of all instances failing is

$$\left(1 - 1/\binom{n}{2k-1}\right)^{O\left(\binom{n}{2k-1} \log(n)\right)} \leq e^{-O(\log n)} = 1/\operatorname{poly}(n).$$

Each instance takes $O\Big(m \log(n) + k^{2k}\Big) \leq O\Big(n^{2k}\Big)$ time if we use a disjoint union data structure to track the contractions. Thus the total running time is

$$O\Big(n^{4k-2} \log(n)\Big).$$

Part 4. An $n^{O(k)}$ running time is not polynomial when k is part of the input.

Rubric.

3 pts. Part 1
- **1 pt.** Run until there are $O(k)$ vertices left.
- **2 pts.** Brute force on remaining graph
  - **1 pt.** Enumerate remaining k-cuts
  - **1 pt.** Directly calculate value of each remaining k-cut.

4 pts. Part 2
- **1 pt.** Proving $\lambda \leq 2(k-1)c(E)/n$.
- **1 pt.** Analyzing probability of surviving a particular round.
- **1 pt.** Chaining probabilities together with conditional probabilities.
- **1 pt.** Calculate total probability of algorithm succeeding.

3 pts. Part 3
- **1 pt.** Repeat until high confidence. (algorithmic description and analysis)
- **1 pt.** Time complexity analysis

1 pt. Part 4
- Note that $n^{O(k)}$ is not polynomial if k is part of the input.

B.3 Homework 2

Exercise C.14. Prove Item 2 of Theorem 5.3.[14]

Solution to exercise C.14. The analysis is similar to that of the gap theorem. For $i \in \mathbb{N}$, let A_i, B_i, and v_i be as in the proof of the gap theorem. As before, we have a connected component of size k iff $|A_i| \geq i$ for all $i \leq k$.

The key idea in the gap theorem is that $|A_i|$ is an independent sum of $\{0,1\}$-indicator variables with $\mathbf{E}[|A_i|] \geq (1 + \Omega(\epsilon))i$, so for $i = \Omega\big(\log(n)/\epsilon^2\big)$, $|A_i| \geq i$ with high probability. The key idea here is similar: we have (roughly) $\mathbf{E}[|A_i|] \leq (1-\epsilon)i$, so for $i = \Omega\big(\log(n)/\epsilon^2\big)$, we have $|A_i| < i$ with high probability.

More precisely, A_i independently samples each vertex (except the starting one) with probability $q_i = 1 - (1-p)^i$. By the union bound, the probability a vertex appears in A_i is bounded above by $q_i \leq pi$. Thus, by linearity of expectation,

$$\mathbf{E}[|A_i|] = 1 + (n-1)q_i \leq 1 + (n-1)pi \leq 1 + (1-\epsilon)i.$$

By the Chernoff bound, for $i = \Theta\big(\log(n)/\epsilon^2\big)$, $|A_i| < i$ with high probability. That is, for a particular starting vertex v, v's connected component has at most $O\big(\log(n)/\epsilon^2\big)$ vertices with high probability. Taking the union bound over all v, we conclude that with high probability, all connected components have at most $O\big(\log(n)/\epsilon^2\big)$ vertices.

Rubric.
2 pts. Setup the analysis

- Should establish is an independent sum of $\{0,1\}$-indicator variables and we have a connected component of size k iff $|A_i| \geq i$ for all $i \leq k$.

4 pts. Analyze $\mathbf{E}[|A_i|] \leq (1 - \Omega(\epsilon))i$.

- Bound the probability of any particular vertex appearing in A_i.
- Apply linearity of expectation over all non-starting vertices.

2 pts. Apply Chernoff bound to show that $|A_i| < i$ with high probability.
2 pts. Union bound over all starting vertices.

[14] It may be helpful to understand the proof of the gap theorem (Lemma 5.4) in Section 5.3.

Exercise C.64. Design and analyze a deterministic 3/4-approximation algorithm for max-SAT.[15]

Solution to exercise C.64.

Recall that a (3/4)-approximation can be obtained (in expectation)by the better of uniform random sampling and randomly rounding the LP. The uniform random sampling algorithm was already derandomized by the method of conditional expectations in Section 1.2.

We can also derandomize the LP rounding algorithm by using the method of conditional expectations. To do this we need to be able to estimate the number of clauses to be satisfied. This is implicit in the analysis. We first compute the LP solution y. Letting A denote the the number of clauses that are satisfied, we have

$$\mathbf{E}[A] = \sum_{i=1}^{m} 1 - \prod_{j:x_j \in C_i} (1 - y_j) \prod_{j:\bar{x}_j \in C_i} y_j .$$

Note that $\mathbf{E}[A]$ can be computed explicitly in polynomial time. By the method of conditional expectations, we can iteratively round each y_j to $\{0, 1\}$-values to produce an assignment $x_1, \dots, x_n \in \{\mathtt{t}, \mathtt{f}\}$ satisfying $\mathbf{E}[A]$ clauses.

Meanwhile, let B be the number of clauses satisfied by a uniformly random assignment. Recall that the uniformly random assignment can be derandomized (again by the method of conditional expectations) so that at least $\mathbf{E}[B]$ clauses are satisfied.

Thus we have determinstic algorithms that satisfied $\mathbf{E}[A]$ and $\mathbf{E}[B]$ clauses, respectively. The analysis from Section 6.1 shows that

$$\max\{\mathbf{E}[A], \mathbf{E}[B]\} \geq \frac{1}{2}(\mathbf{E}[A] + \mathbf{E}[B]) \geq (3/4)\,\mathrm{OPT}\,.$$

Thus we run our two deterministic algorithms and output the one satisfying more assignments to obtain a determinstic (3/4)-approximation.

[15]You may first want to design and analyze a deterministic $(1 - 1/e)$-approximation algorithm for max-SAT.

Rubric.
1 pt. Derandomizing the oblivious algorithm – reference to Chapter 1 is fine.
7 pts. Derandomizing randomized rounding
- **1 pt.** Solving the LP
- **2 pts.** Formulating the expected number of satisfied clauses as a function of the y_j's.
- **1 pt.** Noting that the formula can be calculated.
- **3 pts.** Derandomizing with conditional expectations.

2 pts. Taking the max of the two derandomized algorithms.
- Should point out that this is at least as good as the their average.

Exercise C.45. Consider an instance of (weighted) set cover defined by sets $S_1, \ldots, S_n \subseteq [m]$ and costs $c_i > 0$ for each set S_i. The goal is to compute the minimum cost collection of sets covering $[m]$. We saw that solving the LP and then randomly rounding gives a $O(\log m)$ approximation. Here we consider a special case where all the sets are small and obtain a better approximation factor by a standard extension of randomized rounding called *alterations*.

Let $\Delta \in \mathbb{N}$ be such that $|S_j| \leq \Delta$ for all j. Consider the algorithm `round-and-fix` for which some speudocode is given below. `round-and-fix` is similar to randomized rounding and has two stages. The first stage solves the LP and then *rounds* the solution scaled up by some factor $\alpha \geq 1$. It is possible that some of the elements $i \in [m]$ may not be covered. In the second stage, we *fix* each uncovered element by (deterministically) taking the cheapest set that covers it.

```
round-and-fix(sets $S_1, \ldots, S_n \subseteq [m]$, costs $c \in \mathbb{R}^n_{>0}$, $\alpha \geq 1$)
1. let $x \in [0,1]^n$ solve the set cover LP
2. let $F \subseteq \{S_1, \ldots, S_n\}$ sample each set $S_i$ independently with probability
   $\min\{1, \alpha x_i\}$
3. for each $i \in [m]$
   A. if $i$ is not covered by $F$
      1. add the cheapest set covering $i$ to $F$
4. return $F$
```

Show that for an appropriate choice of α, this algorithm returns a $O(\log \Delta)$ approximation to the set cover instance (in expectation). (It is possible to get $\log \Delta + \log\log \Delta + O(1)$ with care.)

Solution to exercise C.45.

Let $x \in \mathbb{R}^n_{\geq 0}$ be the LP solution. (Here the LP from the lecture is adjusted to account for the set costs, by replacing the objective with $\sum_{i=1}^n c_j x_j$.) For each point i, let $S_{j(i)}$ be the least expensive set covering i. Note that

$$c_{j(i)} \overset{\text{(a)}}{\leq} c_{j(i)} \left(\sum_{j:i \in S_j} x_j \right) \overset{\text{(b)}}{\leq} \sum_{j:i \in S_j} c_j x_j \tag{B.1}$$

where (a) is because of the covering constraint and (b) is because $c_{j(i)} \leq c_j$ for all sets S_j covering i.

When we scale up x by $\alpha = O(\log(\Delta))$ (instead of $\log(m)$) and then randomly round, then we have the guarantee that for all $i \in [m]$, we either αx_j for some set j covering i, or

$$\mathbf{P}[i \text{ not covered}] \leq \prod_{S_j \ni i} (1 - \alpha y_j) \leq e^{-\alpha \sum_{S_j \ni i} y_j} \leq \frac{1}{\Delta^2} \tag{B.2}$$

since $\sum_{S_j \ni i} y_j \geq 1$. If i is not covered, then we pay at most $c_{j(i)}$ to cover it. Thus the expected cost of fixing is bounded above by

$$\begin{aligned}\sum_{i \in [m]} \mathbf{P}[i \text{ not covered}] c_{j(i)} &\overset{\text{(c)}}{\leq} \frac{1}{\Delta^2} \sum_{i \in [m]} c_{j(i)} \overset{\text{(d)}}{\leq} \frac{1}{\Delta^2} \sum_{i \in [m]} \sum_{j:i \in S_j} c_j x_j \\ &\overset{\text{(e)}}{=} \frac{1}{\Delta^2} \sum_{j=1}^n c_j x_j |S_j| \overset{\text{(f)}}{\leq} \frac{1}{\Delta} \sum_{j=1}^n c_j x_j = \frac{1}{\Delta} \mathrm{OPT}_{\mathrm{LP}}.\end{aligned}$$

(c) is by inequality (B.2). (d) is by inequality (B.1). (e) is by interchanging sums. (f) is because each set has size $\leq \Delta$. Altogether we pay

$$\alpha \, \mathrm{OPT}_{\mathrm{LP}} + \frac{1}{\Delta} \mathrm{OPT}_{\mathrm{LP}} \leq O(\log(\Delta)) \, \mathrm{OPT}_{\mathrm{LP}}.$$

Rubric.
2 pts. Scale up by $O(\log(\Delta))$.
3 pts. Probability a particular set is not covered is $1/\operatorname{poly}(\Delta)$.
5 pts. Bounding the expected cost of fixing.

- Initially, bound the fixing cost with the sum, over all elements, of the probability of not covering the element times the minimum cost of any set containing the element.
- Somehow, maybe implicitly, bound the sum of minimum-cost sets by $\Delta \, \mathrm{OPT}_{\mathrm{LP}}$.
- Use $1/\operatorname{poly}(\Delta)$ probability of failing to cover an element to offset the leading Δ factor above.

Exercise C.9. Recall that in the heavy hitters problem, the goal was to estimate the absolute frequency of each element (in $[n]$) up to an additive error of ϵm, where m is the total length of the stream. Another way to frame this to first let $x \in \mathbb{R}^n$ denote the frequency vector; that is, x_i is the absolute frequency of element i, and $\|x\|_1 = m$. We can think of `count-min` as estimating each coordinate x_i with (one-sided) additive error of $\epsilon\|x\|_1$.

In this problem we do something similar except with respect to the ℓ_2-norm. The goal is to estimate each coordinate x_i up to an additive error of $\pm\epsilon\|x\|_2$, and holds for real-valued $x \in \mathbb{R}^n$ (unlike `count-min`, which only holds for nonnegative vectors). Formally, we start with the all-zero vector $x = \mathbb{0}^n$. The stream presents data of the form (i, Δ), where $i \in [n]$ and $\Delta \in \mathbb{R}$, which indicates the update $x_i \leftarrow x_i + \Delta$. We want a data structure that can estimate each coordinate x_i up to $\pm\epsilon\|x\|_2$ with high probability, in sublinear space.

Below we describe a data structure that can get $\pm\epsilon\|x\|_2$ error for an appropriate choice of parameters. (This is like describing one "row" of the `count-min` data structure.) You will first be asked to choose the parameters and prove the error guarantee. Then you will be asked to amplify the data structure to obtain a high probability guarantee.

The data structure is as follows. Let $\epsilon > 0$ be given, let $w \in \mathbb{N}$ be a parameter TBD, and let $A[1..w]$ be an array of values initially set to 0. Let $h : [n] \to [w]$ be a pairwise independent hash function. Let $g : [n] \to \{-1, 1\}$ be a second pairwise independent hash function. The operations are as follows.

- For each update (i, Δ) presented by the stream, we set $A[h(i)] \leftarrow A[h(i)] + g(i)\Delta$.
- To retrieve an estimate for coordinate i, we return $g(i)A[h(i)]$.

We now analyze this approach, as follows.

1. For each i, let $y_i = g(i)A[h(i)]$ denote the estimate returned by the data structure. Prove that y_i is an unbiased of x_i for each i. (That is, $\mathbf{E}[y_i] = x_i$ for all i.)
2. What is the variance of y_i (as a function of w)?
3. Prove that for an appropriate choice of w, the probability that $|x_i - y_i| \geq \epsilon\|x\|_2$ is at most $1/10$. (w should depend on ϵ, and in general, the smaller the choice of w, the better. The choice of $1/10$ is arbitrary; any probability less than $1/2$ would suffice.)
4. Using the data structure designed above, design and analyze a data structure that, in $O(\log(n)/\epsilon^2)$ space, estimates each coordinate of x up to an additive error of $\pm\epsilon\|x\|_2$ with high probability. (I.e., probability of error at most $1/\operatorname{poly}(n)$.)

Solution to exercise C.9. This data structure is called `count-sketch` [CCF02], and is a common topic in algorithms on sketching and streaming [Nel20; Che14]. See [Che14, §2] for the solution.

Part 1. We have

$$\mathbf{E}[y_i] = \mathbf{E}[g(i)A[h(i)]] = \sum_j \mathbf{E}[g(i)g(j)]x_j \, \mathbf{P}[h(i) = h(j)] = x_i$$

since $\mathbf{E}\left[g^2(i)\right] = 1$, and $\mathbf{E}[g(i)g(j)] = 0$ for $i \neq j$ by pairwise independence.

Part 2. We have

$$\begin{aligned}\mathbf{E}\Big[y_i^2\Big] &= \sum_{k=1}^n \sum_{\ell=1}^n \mathbf{E}\Big[g^2(i)g(k)g(\ell)\Big] \, \mathbf{P}[h(i) = h(k) = h(\ell)]x_k x_\ell \\ &\overset{\text{(g)}}{=} \sum_{k=1}^n \mathbf{P}[h(k) = h(\ell)]x_k^2 \\ &\overset{\text{(h)}}{=} x_i^2 + \frac{1}{w}\sum_{k \neq i} x_k^2.\end{aligned}$$

For (g), we observe that

$$\mathbf{E}\Big[g^2(i)g(k)g(\ell)\Big] = \mathbf{E}[g(k)g(\ell)] = \begin{cases} 1 & \text{if } k = \ell \\ 0 & \text{if } k \neq \ell \end{cases}$$

by pairwise independence of g. (h) is by pairwise independence of h.

Now we have

$$\begin{aligned}\mathrm{Var}[y_i] &= \mathbf{E}[y_i]^2 - \mathbf{E}\Big[y_i^2\Big] \\ &= \frac{1}{w}\sum_{k \neq i} x_k^2 \\ &=\leq \frac{1}{w}\|x\|_2^2.\end{aligned}$$

Part 3. By Chebyshev's inequality,

$$\mathbf{P}[|x_i - y_i| \geq \epsilon\|x\|_2] \leq \frac{\mathrm{Var}[y_i]}{\epsilon^2\|x\|_2^2} = \frac{1}{w\epsilon^2}.$$

For $w \geq 10/\epsilon^2$, the RHS is at most $1/10$.

Part 4. We make $\ell = O(\log n)$ copies of the single "row" described above with $w = \lceil 10/\epsilon^2 \rceil$. When retrieving an estimate coordinate for coordinate i, we return the median of the y_i's over all the rows.

Call a single row's estimate y_i "good" if $|x_i - y_i| \leq \epsilon\|x\|_2$ and "bad" otherwise. The median over the y_i's is good if less than half the row-estimates y_i is bad. A single row-estimate y_i is bad with probability at most $1/10$, so we expected $\ell/10$ bad estimates. By the

Chernoff bound, the probability that we get at least $\ell/2$ bad estimates is bounded above by

$$e^{-\Omega(\ell)} = \frac{1}{\text{poly}(n)}.$$

So for each coordinate, the median is good with high probability. Taking the union bound over all coordinates, we get additive error $\pm\epsilon\|x\|_2$ for every coordinate with high probability.

Rubric.

2 pts. Part 1

- Analyze expected value (correctly).
- Should point out where pairwise independence (for g) is used.

2 pts. Part 2

- Correct analysis of variance.
- Note where pairwise independence is used for each hash function.

3 pts. Part 3

- Apply Chebyshev's inequality.
- Deduce $w = O(1/\epsilon^2)$.

3 pts. Part 4

- Duplicate rows $O(\log n)$ times.
- Return median.
- Point out that median is bad if and only if (at least) half the rows are bad.
- Apply Chernoff to show the median is good with high probability.
- Take the union bound over all coordinates.

Appendix C

Scrambled problems

The exercises can be more fun when you don't know which chapter they come from. Here are all the problems, over all chapters covered in class, in random order.

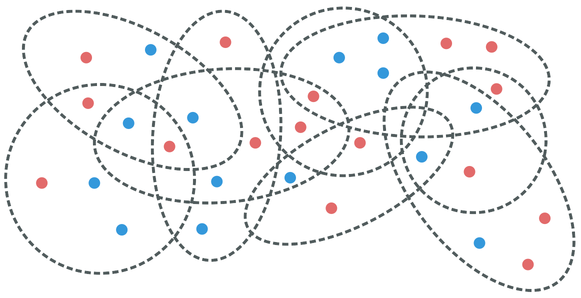
Exercise C.1. Consider the particular case of hash tables with chaining with $k = n$ and an *ideal* hash function $h : [n] \to [n]$. Let $A[1..n]$ be the cells of the hash table.

1. Consider a particular array slot $A[i]$. Show that for $\ell \in \mathbb{N}$, the probability that $A[i]$ has $\geq \ell$ items hashed to it is

$$\mathbf{P}[\text{at least } \ell \text{ items being hashed to } A[i]] \leq \frac{1}{\ell!}.$$

2. Show that, with probability of error $\leq 1/n^2$, the maximum length is at most $O(\log(n)/\log\log n)$.[1]

Exercise C.2. A classical case of *discrepancy* is the following "balanced covering problem". Suppose you have n sets over n points. The goal is to color all the points either red or blue, so that each set has the same number of points of each color, or as close to the same as possible.



[1]The simple lower bound of $\ell! \geq (\ell/2)^{\ell/2}$ may be helpful. It is implicit in the $O(\cdots)$ notation that your bound need only hold for n sufficiently large.

Numerically this is generalized and modeled as follows. Let $A \in [0,1]^{n \times n}$ be a square matrix with bounded entries. For a vector of signs $x \in \{-1,+1\}^n$, the *discrepancy* of x is the quantity

$$\|Ax\|_\infty = \max_i |Ax|_i.$$

(If A is the $\{0,1\}$ incidence matrix of the n sets (as rows) to the n points (as columns), if we let $+1$ denote the color blue -1 denote the color red, then $\|Ax\|_\infty$ is the maximum color imbalance over all the sets.) The general goal is to chose $x \in \{-1,+1\}^n$ to minimize the discrepancy.

Here the goal is to establish a baseline for this problem. Design and analyze an algorithm that computes a point $x \in \{-1,1\}^n$ with discrepancy

$$\|Ax\|_\infty \leq O\left(\sqrt{n \log n}\right).$$

Later in the semester we will improve the bound fto $\|Ax\|_\infty \leq O(\sqrt{n})$.

Exercise C.3. Prove or disprove: for any two random variables X, Y, and real-valued function $f(X,Y)$, we have

$$\mathop{\mathbf{E}}_{X}\left[\mathop{\mathbf{E}}_{Y}[f(X,Y) \mid X]\right] = \mathop{\mathbf{E}}_{Y}\left[\mathop{\mathbf{E}}_{X}[f(X,Y) \mid Y]\right].$$

Exercise C.4. Let $G = (V,E)$ be an undirected graph. For $k \in \mathbb{N}$ a *k-cut* is a set of edges whose removal disconnects the graph into at least k connected components. Note that for $k \geq 3$, the minimum k-cut problem cannot easily be reduced to (s,t)-flow. In fact, the problem is NP-Hard when k is part of the input.[2]

1. Briefly describe how to modify the `random-contractions` to return a k-cut.[3]

2. Analyze the probability that your modified algorithm returns a minimum k-cut.[4]

3. Describe and analyze an algorithm, using your modified `random-contractions` as a subroutine, that computes a minimum k-cut with high probability in $O\left(n^{c_1 k} \log^{c_2} n\right)$ time for constants c_1 and c_2. (We leave it to you to identify these constants; as usual, the faster the running time, the better.)

[2]You might find it helpful to focus on the case $k = 3$ and then generalize afterwards. Although we ask you to work out the dependency on k, conceptually, it might help to think of k as being relatively small compared to n.

[3]Your algorithm design may be informed by your calculations in part 2.

[4]You may want to pattern your analysis after the one for minimum (2-)cut; in particular, you may want to develop analogs for Lemmas 4.2 and 4.3.

4. How does your algorithm relate to the preceding statement that k-cut is NP-Hard when k is part of the input?

Exercise C.5. Recall the set cover problem for which we obtained a randomized $O(\log n)$-approximation. Here we consider a (maximum weight) set *packing* problem, defined as follows.

Let $[m]$ be a set of points, and let $S_1, \dots, S_n \subseteq [m]$ be n subsets of $[m]$. Let $b_1, \dots, b_n > 0$ represent the profit of $S_1, \dots, S_n$, respectively. We say that a collection of sets $\mathcal{F} = \{S_{j_1}, \dots, S_{j_k}\}$ is a *set packing* if they are all disjoint. The total profit of such a set packing is defined as the sum of profits $b_{j_1} + \cdots + b_{j_k}$ of the corresponding sets.

The goal is to compute a set packing of maximum profit, but the problem is NP-Hard. Here we consider the following (perhaps unusual) approximation criteria. Let OPT denote the maximum profit of any set packing. For $\alpha \geq 1$, we say that a collection of sets $S_{j_1} + \cdots + S_{j_k}$ is an α-packing if each point is covered by at most α sets S_{j_h}. We say that a randomized collection of sets $\mathcal{F}$ is a *randomized approximate α-packing* if

1. The expected total profit of $\mathcal{F}$ is at least OPT.
2. With high probability, $\mathcal{F}$ is an α-packing.

Design and analyze a polynomial time algorithm that outputs a randomized approximate α-packing for α as small as possible.[5]

Exercise C.6. Let $h : [n] \to [\ell]$ be an ideal hash function, with $\ell \geq n$. What is the *exact probability* that h has no collisions (i.e., h is injective)?

Exercise C.7. Recall that the following randomized greedy algorithm gets a $1/e$-approximation for maximizing normalized nonnegative submodular functions subject to a cardinality constraint of k.

```
randomized-greedy($f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$, $\mathcal{N}$, $k$)
1. $S_0 \leftarrow \emptyset$.
2. For $i = 1, \dots, k$:
    A. Let $R_i \subseteq \mathcal{N}$ be the $k$ elements $e$ with maximum $f(e \mid S_{i-1})$.
    B. Sample $e_i \sim R_i$ uniformly at random.
    C. Set $S_i \leftarrow S_{i-1} + e_i$.
3. Return $S_k$.
```

[5]Here we are interested in the approximation factor α – the smaller and closer to 1 the better – and not the exact polynomial running time.

What if f was also monotone? Analyze the randomized greedy algorithm for normalized monotone submodular functions f.

Exercise C.8. Let $P \in \mathbb{R}^{\ell \times n}$ be the random projection function as described in Theorem 9.1, for a parameter ℓ to be determined. We want to argue that for $\ell = O(k/\epsilon^2)$, P approximately preserves all of the vectors of a fixed subspace U of dimension k with high probability (in k).

To express this more formally, let U be a fixed (but unknown) subspace of $\mathbb{R}^n$ of dimension k. We claim that, with probability at least $1 - e^{-O(k)}$, we have

$$(1-\epsilon)\|x\|^2 \leq \|Px\|^2 \leq (1+\epsilon)\|x\|^2 \text{ for all } x \in U \text{ (simultaneously).} \tag{C.1}$$

Note that the algorithm does not know U; for this reason, P is called an $(1 \pm \epsilon)$-approximate *oblivious subspace embedding*. Oblivious subspace embeddings are useful for developing faster approximation algorithms in numerical linear algebra.

In this exercise we prove that P is an oblivious subspace embeddings with high probability. Let U be a subspace of dimension k. Let $\mathbb{S}^k$ be the unit sphere in the k-dimensional subspace we want to preserve. Since the requirements of (C.1) are scale invariant, it suffices to establish (C.1) for all points in $\mathbb{S}^k$.

Our argument makes use of a geometric technique called ϵ-*nets*. For a set S, an ϵ-net is a set N such that for every point $s \in S$, there is a point $x \in N$ such that $\|x - s\| \leq \epsilon$.[6] We need to show that there exists a relatively small ϵ-net for $\mathbb{S}^k$.

1. Let $N \subseteq \mathbb{S}^k$ be a maximal set of points such that any two points in N have distance at least ϵ. Show that N is an ϵ-net, and that N has at most $(1 + 2/\epsilon)^k$ points.[7]

Let N be an $(1/2)$-net of $\mathbb{S}^k$ with at most 5^k points. Next we establish that we preserve the length of all points in N is preserved with high probability.

2. Show that with probability $1 - e^{-O(k)}$, we have $\left|\|Px\|^2 - 1\right| \leq \epsilon$ for all $x \in N$, and $|\langle Px, Py\rangle - \langle x, y\rangle| \leq \epsilon$ for all $x, y \in N$.

So now we know that we preserve all points of N with high probability. We want to argue that this suffices to preserve all the vectors.

3. Prove that for any unit vector $x \in \mathbb{S}^k$, one can write $x = x_0 + x_1 + x_2 + \cdots$ such that for all i:

   (a) $\|x_i\| \leq 2^{-i}$.

[6] This is similar but different from the ϵ-nets in Chapter 13.

[7] *Hint:* N packs $|N|$ interior-disjoint k-dimensional balls of radius $\epsilon/2$ into an k-dimensional ball of radius $1 + \epsilon/2$. It is helpful to know that the volume of an n-dimensional ball of radius r is $c_n r^n$ for a parameter $c_n > 0$ depending on n.

(b) $x_i/\|x_i\| \in N$.[8]

Now use the representation above to prove that P is an oblivious subspace embedding.

4. Prove that (C.1) holds with probability at least $1 - e^{-O(k)}$.

The high-level takeaway from the proof is that if you can embed an ϵ-net of the unit sphere for constant ϵ, then you automatically embed the entire subspace.

Exercise C.9. Recall that in the heavy hitters problem, the goal was to estimate the absolute frequency of each element (in $[n]$) up to an additive error of ϵm, where m is the total length of the stream. Another way to frame this to first let $x \in \mathbb{R}^n$ denote the frequency vector; that is, x_i is the absolute frequency of element i, and $\|x\|_1 = m$. We can think of `count-min` as estimating each coordinate x_i with (one-sided) additive error of $\epsilon\|x\|_1$.

In this problem we do something similar except with respect to the ℓ_2-norm. The goal is to estimate each coordinate x_i up to an additive error of $\pm\epsilon\|x\|_2$, and holds for real-valued $x \in \mathbb{R}^n$ (unlike `count-min`, which only holds for nonnegative vectors). Formally, we start with the all-zero vector $x = \mathbb{0}^n$. The stream presents data of the form (i, Δ), where $i \in [n]$ and $\Delta \in \mathbb{R}$, which indicates the update $x_i \leftarrow x_i + \Delta$. We want a data structure that can estimate each coordinate x_i up to $\pm\epsilon\|x\|_2$ with high probability, in sublinear space.

Below we describe a data structure that can get $\pm\epsilon\|x\|_2$ error for an appropriate choice of parameters. (This is like describing one "row" of the `count-min` data structure.) You will first be asked to choose the parameters and prove the error guarantee. Then you will be asked to amplify the data structure to obtain a high probability guarantee.

The data structure is as follows. Let $\epsilon > 0$ be given, let $w \in \mathbb{N}$ be a parameter TBD, and let $A[1..w]$ be an array of values initially set to 0. Let $h : [n] \to [w]$ be a pairwise independent hash function. Let $g : [n] \to \{-1, 1\}$ be a second pairwise independent hash function. The operations are as follows.

- For each update (i, Δ) presented by the stream, we set $A[h(i)] \leftarrow A[h(i)] + g(i)\Delta$.
- To retrieve an estimate for coordinate i, we return $g(i)A[h(i)]$.

We now analyze this approach, as follows.

1. For each i, let $y_i = g(i)A[h(i)]$ denote the estimate returned by the data structure. Prove that y_i is an unbiased of x_i for each i. (That is, $\mathbf{E}[y_i] = x_i$ for all i.)

2. What is the variance of y_i (as a function of w)?

[8]Hint: Choose x_0 to be the closet point in N to x. Observe that $\|x - x_0\| \leq 1/2$ because N is a (1/2)-net. It remains to express $x - x_0$ as the sum $x_1 + x_2 + \cdots$. How might you choose x_1?

3. Prove that for an appropriate choice of w, the probability that $|x_i - y_i| \geq \epsilon \|x\|_2$ is at most 1/10. (w should depend on ϵ, and in general, the smaller the choice of w, the better. The choice of 1/10 is arbitrary; any probability less than 1/2 would suffice.)

4. Using the data structure designed above, design and analyze a data structure that, in $O(\log(n)/\epsilon^2)$ space, estimates each coordinate of x up to an additive error of $\pm\epsilon\|x\|_2$ with high probability. (I.e., probability of error at most $1/\operatorname{poly}(n)$.)

Exercise C.10. Prove Turán's theorem:

> **Turan's theorem***Any undirected graph G with m edges and n vertices has an independent set of size (at least)*
>
> $$\frac{n^2}{2m+n}.$$

(You may want to solve Exercise C.70 first.)

Exercise C.11. Let P be a set of n distinct points in the plane. Consider the problem of computing the minimum distance $\|p - q\|$ between any two distinct points $p, q \in P$. (If $|P| \leq 1$, the minimum distance is defined as $+\infty$.) For simplicity we assume the points are in general position, so that all pairwise distances are unique. You may have learned a deterministic divide-and-conquer algorithm that runs in $O(n \log n)$ time. Here we will analyze the running time of a different randomized algorithm. To help build intuition, first solve the following problem.

1. (3 points) Let $a_1, \ldots, a_n \in \mathbb{R}$ be n distinct numbers presented in a uniformly random order. Imagine tracking the minimum as you examine the numbers one by one in (the randomized) order. What is the expected number of times the minimum changes?

We assume black box access to a data structure for the following "threshold" version of the minimum distance problem. Fix a threshold $\rho > 0$. The data structure takes $O(1)$ time to initialize, and starts with an empty point set. It supports the following operation:

> `insert(`x`)`: Insert a point $x \in \mathbb{R}^2$ into the underlying point set. If the minimum distance in the underlying point set is (strictly) less than ρ, then return `true`; otherwise return `false`. This operation takes $O(1)$ expected time.[9]

[9]We can implement this data structure by building a "hash table over grid cells". Initially we have an empty dictionary for an empty point set. Given a point $x = (x_1, x_2)$, compute the key

This subroutine, combined with part 1, suggests the following algorithm. Here we assume that one can do math operations (add, subtract, square, etc.) in $O(1)$ time, and that we can sample a random permutation of n items in $O(n)$ time.

```
1. Let $p_1, p_2, \ldots, p_n$ permute $P$ uniformly at random.        // $O(n)$ time.
2. Let $r_1 = +\infty$ and $r_2 = \|p_1 - p_2\|$.
/* We maintain the invariant that $r_i$ is the minimum distance over $p_1, \ldots, p_i$. */
3. Start a new instance of the threshold data structure with threshold $r_2$. Insert
   $p_1$ and $p_2$.
4. For $i = 3, \ldots, n$:
   A. Insert $p_i$ into the threshold data structure. If the data structure returns
      true:
      1. Let $r_i = \min\{\|p_i - p_j\| : j < i\}$.
      2. Replace the threshold data structure with a new one with radius $r_i$.
         Insert $p_1, \ldots, p_i$ into the data structure.
   B. Otherwise set $r_i = r_{i-1}$.
5. Return $r_n$.
```

Now analyze this algorithm via the following steps.

2. (3 points) For each index $i > 2$, analyze the exact probability that $r_i < r_{i-1}$.

3. (4 points) Finally, bound the expected running time of the algorithm.

Exercise C.12. Let $P \subseteq \mathbb{R}^d$ be a set of n points. Let $f : \mathbb{R}^d \to \mathbb{R}^k$ be a random projection with $k = O(\log(n)/\epsilon^2)$ (per Theorem 9.1). Recall that with high probability (say, $\geq 1 - 1/n^4$), we have

$$(1-\epsilon)\|x\|^2 \leq \|f(x)\|^2 \leq (1+\epsilon)\|x\|^2$$

for all $x \in P$, and we also have

$$(1-\epsilon)\|x-y\|^2 \leq \|f(x) - f(y)\|^2 \leq (1+\epsilon)\|x-y\|^2$$

`key(`x`)` $= (\lfloor x_1/(\rho/\sqrt{2})\rfloor, \lfloor x_2/(\rho/\sqrt{2})\rfloor$ in $O(1)$ time. (Rounding is not a standard operation in all models of computation, but here we assume it takes $O(1)$ time.) If this key is already occupied by a point $y \in P$, then $\|x - y\| \leq \rho$, and we return `true`. Otherwise consider the 8 "neighboring" keys/cells where we add or subtract at most one or zero to each of the key's coordinates. Assuming the minimum distance was $\geq \rho$ before inserting x, each of these 8 cells can have at most 1 point. If the distance between x and any of the $O(1)$ points in neighboring cells is $< \rho$ from x, return `true`.

as well as

$$(1-\epsilon)\|x+y\|^2 \le \|f(x)+f(y)\|^2 \le (1+\epsilon)\|x+y\|^2$$

for all $x, y \in P$. Show that with high probability, we also have

$$|\langle f(x), f(y)\rangle - \langle x, y\rangle| \le \epsilon\|x\|\|y\|$$

for all $x, y \in P$.[10]

Exercise C.13. Prove that $\mathrm{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$.

Exercise C.14. Prove Item 2 of Theorem 5.3.[11]

Exercise C.15. In CIPs we allowed each x_j to be as large as we want. Suppose we added the constraint $x_j \le 1$ for all j. Would the randomized rounding algorithm from Section 6.3 still obtain a $(1-1/e)$-approximation ratio? Why or why not?

Exercise C.16. Let $\ell < k$. Prove that an LRU cache of size k is $k/(k+1-\ell)$-competitive with any cache of size ℓ.

That means, for example, that an LRU cache of size k is 2-competive with any cache of size $k/2$. Some people find this bound more compelling.

You should be able to prove this by a short modification of the argument of the k-competitive bound. It suffices to point out which part of the argument should change, and how.

Exercise C.17. This exercise is about estimating F_k for $k > 2$ with sublinear space as the elements are presenting in a stream (as described in the introduction). Let $g(x) = x^k$ for fixed x. Then $F_k = \sum_i g(f_i)$, where f_i is the frequency of item i. We break down the design and analysis of such an estimator into steps below, but you are encouraged to try to design one yourself first.

For each element $e \in [n]$, and each index in the stream $i \in [m]$, let $f_e^{(i)}$ be the frequency of element i after ith iterations. Consider the random variable Y defined by

$$Y \overset{\text{def}}{=} m\Big(g(f_{e_i}^{(m)} - f_{e_i}^{(i)}) - g(f_{e_i}^{(m)} - f_{e_i}^{(i)} - 1)\Big)$$

where $i \in [m]$ is drawn uniformly at random.

[10] It might by helpful to work through the special case where $\|x\| = \|y\| = 1$. Showing $O(\epsilon\|x\|\|y\|)$ error is fine; a proper $\epsilon\|x\|\|y\|$ then follows from dividing ϵ by a constant factor.

[11] It may be helpful to understand the proof of the gap theorem (Lemma 5.4) in Section 5.3.

1. How can you implement Y in a streaming fashion? (In a stream, you don't know m).[12]

2. Prove that Y is an unbiased estimator for F_k.

3. Prove that the variance of Y is at most $kn^{1-1/k}F_k^2$.[13]

4. Using Y, design and analyze a streaming algorithm that computes a $(1+\epsilon)$-approximation for F_k with probability $1-\delta$, for given parameters ϵ and δ. In addition to the correctness, analyze the time and space of your algorithm.

Exercise C.18. Show that there exists universal constants $c_1, c_2 > 0$ such that for all x with $|x| \leq c_1$,

$$1 + x \leq e^{x - c_2 x^2}.$$

(In other words, you can choose whatever constants c_1 and c_2 are convenient to you.)

Exercise C.19. This exercise is about a simple randomized algorithm for *verifying* matrix multiplication. Suppose we have three $n \times n$ matrices A, B, C. We want to verify if $AB = C$. Of course one could compute the product AB and compare it entrywise to C. But multiplying matrices is slow: the straightforward approach takes $O(n^3)$ time and there are (more theoretical) algorithms with running time roughly $O(n^{2.37\cdots})$. We want to test if $AB = C$ in closer to n^2 time.

The algorithm we analyze is very simple. Select a point $x \in \{0,1\}^n$ uniformly at random. (That is, each $x_i \in \{0,1\}$ is an independently sampled bit.) Compute $A(Bx)$ and Cx, and compare their entries. (Note that it is much faster to compute $A(Bx)$ than AB.) If they are unequal, then certainly $AB \neq C$ and we output `false`. Otherwise we output `true`. Note that the algorithm is always correct if $AB = C$, but could be wrong when $AB \neq C$. We will show that if $AB \neq C$, the algorithm is correct with probability at last $1/2$.

1. Let $y \in \mathbb{R}^n$ be a fixed nonzero vector, and let $x \in \{0,1\}^n$ be drawn uniformly at random. Show that $\langle x, y \rangle \overset{\text{def}}{=} \sum_{i=1}^n x_i y_i \neq 0$ with probability at least $1/2$. [14]

2. Use the preceding result to show that if $AB \neq C$, then with probability at least $1/2$, $ABx \neq Cx$.[15]

[12]*Hint:* Exercise 2.5.

[13]*Hint:* This is a bit messy. One approach is to first show that $\mathrm{Var}[Y] \leq kF_1F_{2k-1}$, and then bound $F_1F_{2k-1} \leq n^{1-1/k}F_k^2$.

[14]*Hint: Suppose for simplicity that the last coordinate of y is nonzero. It might help to imagine sampling the first $n-1$ bits and computing the partial sum $S_{n-1} = \sum_{i=1}^{n-1} x_i y_i$ first, before sampling x_n and adding $x_n y_n$. Formally your analysis may involve some conditional probabilities. (And what about the case where $y_n = 0$?)*

[15]Even if you haven't solved part 1 you may assume it to be true.

3. Suppose we want to decrease our probability of error to (say) $1/n^2$. Based on the algorithm above, design and analyze a fast randomized algorithm with the following guarantees.

   - If $AB = C$, then it always reports that $AB = C$.
   - If $AB \neq C$, then with probability of error at most $1/n^2$, it reports that $AB \neq C$.

   (Your analysis should include the running time as well.)

Exercise C.20. In this exercise, we will develop a 2-approximate LSH scheme for bit strings $s \in \{0,1\}^d$ of a fixed length d with respect to *Hamming distance*. The Hamming distance between two strings $s, t \in \{0,1\}^d$ this fraction of coordinates in which they differ:

$$\text{Hamming}(s,t) = \frac{|\{i \in [d] : s_i \neq t_i\}|}{d}.$$

Of course one can treat bit strings as vectors in $\mathbb{R}^d$ where the Hamming distance coincides with the Euclidean distance squared. Here we explore an alternative approach.

1. Consider the randomly constructed hash function $h : \{0,1\}^d \to \{0,1\}$ defined by

   $$h(x) = x_i,$$

   where $i \in [d]$ is sampled uniformly at random. For two points $s, t \in \{0,1\}^d$, what is $\mathbf{P}[h(s) = h(t)]$, as a function of the Hamming distance between s and t?

2. Fix a target distance $r \in [0,1]$. Construct a data structure that over a set P of n strings $\{0,1\}^d$ to answer the following query with high probability.

   > *Given a query point $s \in \{0,1\}^d$, either return a point $x \in P$ with Hamming distance $\leq 2r$ from s, or declare that there are no points within Hamming distance r from s.*

   In addition to describing the algorithm, one should analyze the preprocessing time and space, the query time, and the probability of correctness.

3. Briefly describe how to use the above data structure to efficiently find 2-approximate nearest neighbors with respect to Hamming distance (with high probability). Analyze your running time.

Exercise C.21. Recall the randomized $O(\log n)$ approximation algorithm for sparsest cut based on L_1-embeddings. Note that this is also logarithmic in the number of demand pairs, $\binom{n}{2}$. We consider the case where the demands are *sparse*; more precisely, where there are at most k commodities (i.e., pairs) $\{s,t\}$ with nonzero demand $b(s,t) > 0$.

1. Prove the following extension of Theorem 11.14:

   Let $d : V \times V \to \mathbb{R}_{\geq 0}$, and let $T \subseteq V$ be a subset of points with $\ell = |T|$. Then for $h = O\left(\log^2 \ell\right)$, one can compute a randomized embedding $f : V \to \mathbb{R}^h$ such that:

   (a) For all $u, v \in V$, $\|f(u) - f(v)\|_1 \leq O(\log \ell) d(u, v)$ (always).

   (b) With high probability, for all $s, t \in T$, $\|f(s) - f(t)\|_1 \geq d(s, t)$.

2. Describe (rigorously) how to adjust the algorithm and analysis from Section 11.3 to obtain a randomized $O(\log k)$ approximation factor, where k is the number of commodities with nonzero demand.

Exercise C.22. Consider the minimum cut problem in undirected graphs. For $\alpha \geq 1$, we say that a cut $C = \delta(S)$ is an *α-approximate minimum cut* it its capacity is at most α times the capacity of the minimum cut.

1. Let C be an α-approximate minimum cut. Suppose we run the `random-contractions` algorithm run until there are $O(\alpha)$ vertices remaining. Show that C is preserved by the algorithm with probability $\geq 1/\binom{n}{O(\alpha)}$.

2. Show the number of α-approximate minimum cuts is at most

$$n^{O(\alpha)}.$$

Exercise C.23. Extend the approximation algorithm for set cover to positive costs. For each set, there is a positive cost $c_j > 0$. The goal is to compute the minimum cost collection of sets that covers all the points.

Exercise C.24. Suppose you only have access to a coin that flips heads with a known probability p, and tails with (remaining) probability $1 - p$. Describe and analyze a protocol that uses a limited number of tosses of this biased coin in expectation (the smaller the better) to simulate 1 coin toss of a fair coin. (The expected number of biased coin tosses you make may depend on p.)

Exercise C.25. Show how to use the randomized tree metric to randomly round the sparsest cut LP and obtain a $O(\log n)$-approximation for sparsest cut.[16]

[16]Try to prove tree metrics are also L_1-metrics.

Exercise C.26. Prove or disprove: For any two events A, B,

$$\mathbf{P}[A \wedge B] \leq \min\{\mathbf{P}[A], \mathbf{P}[B]\}.$$

Exercise C.27. You run a double-secret laboratory and for your experiments you need to monitor the temperature of the lab very carefully. To this end you can buy thermostats $T_1, \dots, T_k$ that purport to measure the temperature μ, but the thermostats are imperfect. You have the following facts.

1. Given the actual temperature μ of the lab, the thermostat readings T_i are independent.
2. Each thermostat is calibrated so that its expected value $\mathbf{E}[T_i]$ equals the actual temperature μ of the lab.
3. For each thermostat T_i, the variance $\mathrm{Var}[T_i]$ of the thermostat is σ^2 for a known parameter $\sigma > 0$.

Given parameters $\epsilon, \delta \in (0, 1)$, the goal is to be able to measure the temperature of the room with additive error at most ϵ with probability at least $1 - \delta$. Describe and analyze a system that, using as few thermostats as possible[17], obtains additive error ϵ with probability at least $1 - \delta$.

Exercise C.28. Recall the randomized rounding based proof of the max-flow min-cut theorem. Recall that we analyzed a random cut which was based on a threshold $\theta \in (0, 1)$ chosen uniformly at random. Prove or disprove that for (essentially) *all* $\theta \in (0, 1)$, the corresponding cut is a minimum $\{s, t\}$-cut.

Exercise C.29. Recall the bicriteria approximation algorithm for the minimum bisection problem from Section 11.4.

1. Show that the algorithm returns a 1/3-balanced cut.
2. For each iteration i, w/r/t the graph remaining at iteration i, we have
$$\frac{c(\delta(S_i))}{|S_i|} \leq O(\log(n)) \frac{\mathrm{OPT}}{n}.$$
3. Combine the two parts above to prove that the algorithm returns a 1/3-balanced cut of size $O(\log n)\,\mathrm{OPT}$.

[17] up to constant factors independent of ϵ, δ, and σ

Exercise C.30. The deterministic greedy algorithm takes $O(nkQ)$ time where Q denotes an evaluation query to f. For large values of k this may be prohibitively slow. The following algorithm introduces random sampling to try to speed up the algorithm.

```
subsampled-greedy($f$,$\mathcal{N}$,$k$,$\epsilon \in (0,1)$)
1. $S_0 \leftarrow \emptyset$.
2. For $i = 1, \dots, k$:
    A. Let $R \subseteq \mathcal{N}$ sample $n \log(1/\epsilon)/k$ elements independently with repetition.
    B. Let $e_i \in R$ maximize $f(e \mid S_{i-1})$.
    C. Set $S_i \leftarrow S_{i-1} + e_i$.
3. Return $S_k$.
```

Analyze `subsampled-greedy` per the following steps.

1. Analyze the expected running time of `subsampled-greedy`.
2. Prove that for all i, conditional on S_{i-1},

$$\mathbf{E}[f(S_i \mid S_{i-1})] \geq \frac{1-\epsilon}{k}(\mathrm{OPT} - f(S_{i-1})).$$

3. Analyze the overall approximation ratio (in expectation) of S_k.

Exercise C.31. Recall the `quick-select` algorithm introduced in Section 1.1.3. The goal of this exercise is to prove that `quick-select` takes $O(n)$ time in expectation. Below we present two different approaches which offer two different perspectives. Both analyses should use linearity of expectation and we ask you to point this out explicitly.

1. *Approach 1.* Analyze `quick-select` similarly to `quick-sort`, based on the sum of indicators X_{ij}.

   One approach is to reduce to a separate analysis for each of the following 4 classes of pairs:

   (a) X_{ij} where $i < j < k$,
   (b) X_{ij} where $i < k < j$,
   (c) X_{ij} where $k < i < j$, and
   (d) X_{ij} where either $i = k$ or $j = k$.

   For each case, show that the expected sum is $O(n)$.

2. *Approach 2.* The following approach can be interpreted as a randomized divide and conquer argument. We are arguing that with constant probability, we decrease the input by a constant factor, from which the fast (expected) running time follows.

   (a) Consider again `quick-select`. Consider a single iteration where we pick a pivot uniformly at random and throw out some elements. Prove that with some constant probability p, we either sample the kth element or throw out at least $1/4$ of the remaining elements.

   (b) For each integer i, prove that the expected number of iterations (i.e., rounds of choosing a pivot) of `quick-select`, where the number of elements remaining is in the range $[(4/3)^i, (4/3)^{i+1})$, is $O(1)$.[18]

   (c) Fix an integer i, and consider the amount of time spent by `quick-select` while the number of elements remaining is greater than $(4/3)^{i-1}$ and at most $(4/3)^i$. Show that that the expected amount of time is $\leq O\big((4/3)^i\big)$

   (d) Finally, use the preceding part to show that the expected running time of `quick-select` is $O(n)$.

Exercise C.32. Consider the special case of the distinct elements streaming problem where there are $n+1$ total items in a stream, each of which is one of n different possible items $\{1,\dots,n\}$. Show that any algorithm that maintains the number of distinct elements exactly throughout the stream has to use at least n bits of memory.

Exercise C.33. In most textbooks, max flow is presented as the following LP, which in particular has polynomial size in the input graph G.

$$\begin{aligned} \text{maximize } & \sum_{e \in \delta^+(s)} z_e - \sum_{e \in \delta^-(s)} z_e \text{ over } z : E \to \mathbb{R}_{\geq 0} \\ \text{s.t. } & z_e \leq c(e) \text{ for all } e \in E \\ & \sum_{e \in \delta^+(v)} z_e = \sum_{e \in \delta^-(v)} z_e \text{ for all } v \in V \setminus \{s,t\}. \end{aligned} \tag{C.2}$$

The second set of constraints are called *flow conservation* constraints. Show that the above LP is equivalent to the (fractional path packing version of) (Max-Flow) in the following sense.

1. Show that for every (feasible) fractional path packing P, there is a feasible solution z to (C.2) with the same objective value.

[18] Hint: Exercise C.35.

2. Show that for every feasible solution z to (C.2), there is a feasible path packing P with the same objective value.[19]

Exercise C.34. Design and analyze an algorithm that computes a spanning tree with uniform stretch $n-1$ (matching the lower bound induced by the cycle).

Exercise C.35. Suppose you repeatedly flip a coin that is heads with fixed probability $p \in (0,1)$.

1. What is the expected number of coin flips until you obtain one heads?[20] Prove your answer.[21]
2. What is the expected number of coin flips until you obtain two heads? Prove your answer.
3. For general $k \in \mathbb{N}$, what is the expected number of coin tosses until you obtain k heads? Prove your answer.

Exercise C.36. Prove Lemma 5.2. (Here the important part is not the constant, $1/3$ – any constant $c > 0$ is already interesting.)

Exercise C.37. Let A and B be two events with $\mathbf{P}[A] + \mathbf{P}[B] = 1$. Prove or disprove: $\mathbf{P}[A \vee B] = 1$ iff $\mathbf{P}[A \wedge B] = 0$.

Exercise C.38. This exercise is about how for many intents and purposes, we approximately have the extremely convenient identity, "$1 + x \approx e^x$".

1. Prove that for all $x \in \mathbb{R}$, $1 + x \leq e^x$.

   Hint: At $x = 0$, both sides are equal. What are their respective rates of change moving away from 0?

2. Prove that for all $x \leq 1$, $e^x \leq 1 + x + x^2$.

[19] The second problem is trickier than the first. One should be able to prove it using only the ideas and results in this chapter (without retracing the flow algorithms of ensuing chapters).

[20] If the first toss is heads, that counts as one coin flip. If the first toss is tails and the second toss is heads, that counts as two coin tosses. Etc. It may be helpful to first think about a fair coin, where $p = 1/2$.

[21] *Hint:* There is an easy way and a hard way to solve this problem.

Exercise C.39. Prove that given k independent copies $X_1, \dots, X_k$ of the same random variable X,

$$\operatorname{Var}\left[\frac{1}{k}\sum_{i=1}^{k} X_i\right] = \frac{1}{k}\operatorname{Var}[X].$$

Exercise C.40. Complete the proof of Theorem 6.4.

Exercise C.41. The definition of PAC learning requires finding a low-error hypothesis with probability at least $1-\delta$ for aribtrary $\delta > 0$. Consider the following weaker definition of PAC learning where we drop the requirement on δ: let us say that an algorithm is a *weak PAC learner* if for any $\epsilon > 0$, with a training set of size $\operatorname{poly}(1/\epsilon)$, and in randomized polynomial time, it produces a hypothesis with error at most ϵ with probability at least $1/2$. Design and analyze a system that takes as input a weak PAC learning algorithm and produces a PAC learning algorithm (in the original sense).

Exercise C.42. For CIPs, we could assume without loss of generality that $A_{ij} \leq b_j$ for all i, j. (In fact this assumption was critical for applying the Chernoff bound.) Suppose now that we had $\lambda A_{ij} \leq b_j$ for all i, j for a parameter $1 \leq \lambda \leq \log(n)$. Design and analyzing a $O(\log(m)/\lambda)$-approximation algorithm for this setting.

Exercise C.43. The defining characteristic of LPs is that the objective and all linear constraints are given by linear functions. It is natural to generalize this notion and consider mathematical programs where the objective and linear constraints are all given by low-degree polynomials; say, bounded by a degree d. Let us call these "degree d polynomial programs". Linear programs are degree 1 polynomial programs.

Prove that degree d polynomial programs are NP-Hard to solve for $d \geq 3$. To this end, pick a suitable NP-Hard problem, and design a degree 3 polynomial program that can be rounded to a discrete solution without any loss.[22]

Exercise C.44. Our department has a large catalog of classes and a handful of course requirements you have to satisfy to graduate. The requirements seem to be particularly complicated for undergraduates. Each requirement is defined by a subset of classes and a number specifying the number of courses you have to take from this subset.

Formally, let the courses be indexed 1 through n. We have L requirements $(S_1, k_1), \dots, (S_L, k_L)$, each consisting of a set $S_i \subseteq [n]$ and an integer $k_i \in \mathbb{N}$. For each i, you have to take at least k_i classes from S_i. Let $T \subseteq [n]$ be the set of classes you've already taken. Let $m = \sum_i |S_i|$ denote the total size of all the sets.

[22]Better yet, prove the same for $d \geq 2$.

Now we have two possible interpretations of the rules. In the first version, a single class can only count towards a single requirement. In the second version, a single class can count towards any number of requirements. For example, suppose you are required to take ($k_1 = 2$) classes from $S_1 = \{A, B, C\}$ and ($k_2 = 2$) classes from $S_2 = \{C, D, E\}$, and you have already taken $T = \{B, C, D\}$. In the first version, you do not have enough classes to graduate; in the second version, you do have enough classes to graduate.

Here we have related but slightly different problems for the two systems.

1. (5 points) Suppose a single class can only count towards a single requirement. Consider the problem of deciding if you have already taken enough classes to graduate. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.
2. (5 points) Now suppose a single class can count towards any number of requirements. Consider the problem of identifying the minimum number of additional classes that need to be taken to satisfy all the requirements. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

Exercise C.45. Consider an instance of (weighted) set cover defined by sets $S_1, \ldots, S_n \subseteq [m]$ and costs $c_i > 0$ for each set S_i. The goal is to compute the minimum cost collection of sets covering $[m]$. We saw that solving the LP and then randomly rounding gives a $O(\log m)$ approximation. Here we consider a special case where all the sets are small and obtain a better approximation factor by a standard extension of randomized rounding called *alterations*.

Let $\Delta \in \mathbb{N}$ be such that $|S_j| \leq \Delta$ for all j. Consider the algorithm `round-and-fix` for which some speudocode is given below. `round-and-fix` is similar to randomized rounding and has two stages. The first stage solves the LP and then *rounds* the solution scaled up by some factor $\alpha \geq 1$. It is possible that some of the elements $i \in [m]$ may not be covered. In the second stage, we *fix* each uncovered element by (deterministically) taking the cheapest set that covers it.

```
round-and-fix(sets $S_1, \ldots, S_n \subseteq [m]$, costs $c \in \mathbb{R}^n_{>0}$, $\alpha \geq 1$)
1. let $x \in [0,1]^n$ solve the set cover LP
2. let $F \subseteq \{S_1, \ldots, S_n\}$ sample each set $S_i$ independently with probability
   $\min\{1, \alpha x_i\}$
3. for each $i \in [m]$
   A. if $i$ is not covered by $F$
      1. add the cheapest set covering $i$ to $F$
4. return $F$
```

Show that for an appropriate choice of α, this algorithm returns a $O(\log \Delta)$ approximation to the set cover instance (in expectation). (It is possible to get $\log \Delta + \log\log \Delta + O(1)$ with care.)

Exercise C.46. Prove linearity of expectation (Theorem 0.3).

Exercise C.47. We consider the problems of estimating the mean and the median of a stream of numbers. The input consists of a stream of integers $x_1, x_2, \ldots,$ presented one at a time, of unknown length m. Your algorithms should use sublinear space and return $(1 \pm \epsilon)$-approximations (as defined below) of the desired statistics with probability at least $1 - \delta$.

We expect you to analyze the space, the running time, and the probability of outputting an accurate solution. The smaller the space, and in general the lower the dependency on ϵ and δ, the better. For simplicity you can assume it takes $O(1)$ space to store a number and it takes constant time for basic operations like adding, subtracting, and multiplying numbers.[23]

It may be helpful to be aware of a streaming algorithm called *reservoir sampling* that maintains a sample of 1 element drawn uniformly at random from the stream. This algorithm is easy to describe. It takes the first element deterministically. For $i > 1$, with probability $1/i$, it takes the ith element and replaces the element previously held by the algorithm. It is known (and you can assume as fact) that at any point in time, the element held by the algorithm represents an element sampled uniformly at random from the stream. Note that for any $k \in \mathbb{N}$, one can maintain a random sample of k elements with repetition from the stream by running k copies of the single-element reservoir sampling algorithm in parallel.

The two statistics we are interested in are defined as follows. Let $\delta, \epsilon \in (0, 1)$ be fixed.

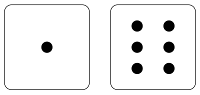
1. (2 points) A $(1 \pm \epsilon)$-approximation of the mean of the stream with probability at least $1 - \delta$. That is, if the mean is x, then you should return a value y in the interval $(1 - \epsilon)x \leq y \leq (1 + \epsilon)x$.

2. (8 points) A rank-wise $(1 \pm \epsilon)$-approximation of the median number in the stream with probability at least $1 - \delta$. That is, if the stream has m total numbers, then you should return an element whose rank r is in the interval $\lfloor (1 - \epsilon)m/2 \rfloor \leq r \leq \lceil (1 + \epsilon)m/2 \rceil$.[24]

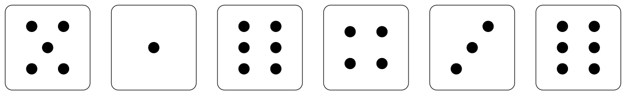
Exercise C.48. Recall that when we roll six-sided dice, the dice samples an integer between 1 and 6 uniformly at random. Let us call an unordered pair of dice "lucky" if one of them is a 1 and the other is a 6.

[23]Of course you are not allowed to abuse this with something bizarre like using a polynomial-bit integer as a bit map. This isn't battlecode.

[24]An element has rank i if it is the ith smallest element.



If we roll 6 independent six-sided dice, how many lucky pairs do we expect? Note that a single dice may appear in more than one lucky pair. For example, the following roll of six dice has 2 lucky pairs amongst them.



Exercise C.49. Prove or disprove: For any two events A and B, if $\mathbf{P}[A] + \mathbf{P}[B] > 1$, then $\mathbf{P}[A \wedge B] > 0$.

Exercise C.50. Provide directly (and without leveraging the techniques from Section 13.2) that sampling $O(\log(g(|P|))/\epsilon)$ points gives an ϵ-net with constant probability.

Exercise C.51. Consider the minimum cut problem in undirected graphs. We say that a cut $C = \delta(S)$ is a 2-*approximate minimum cut* it its capacity is at most twice the capacity of the minimum cut.

1. Let C be an 2-approximate minimum cut. Suppose we run the `random-contractions` algorithm run until there are 5 vertices. Show that C is preserved by the algorithm with probability $\geq 1/\binom{n}{4}$.
2. Show the number of 2-approximate minimum cuts is at most
$$O\Big(n^4\Big).$$

Exercise C.52. Consider the randomized algorithm for minimum cut based on building the minimum spanning tree w/r/t randomized weights, described in Section 4.1.

1. Prove that this algorithm is equivalent to the random contractions algorithm for unweighted graphs.
2. Adjust the randomized spanning tree algorithm to account for weights, and prove its correctness.

Exercise C.53. This exercise asks you to prove the ϵ-net theorem, Theorem 13.4. One can prove it similarly to the ϵ-sample theorem, Theorem 13.2. Below we define the events A and B for you to get you started, and ask you to complete the proof (in full detail).

Let Q_1 and Q_2 be two random samples of points (of appropriate size, TBD by you) inducing measures μ_1 and μ_2, respectively. Define the events A and B by:

$$\begin{aligned} A &\overset{\text{def}}{=} \text{ the event that } Q_1 \cap r = \emptyset \text{ and } \mu(r) \geq \epsilon \text{ for some } r \in R. \\ B &\overset{\text{def}}{=} \text{ the event that } Q_1 \cap r = \emptyset \text{ and } \mu_2(r) > \epsilon/2 \text{ for some } r \in R. \end{aligned}$$

Exercise C.54. Let A and B two events. Prove or disprove that A and B are independent iff $\bar{A}$ and $\bar{B}$ are independent.

Exercise C.55. Let A and B be two events. Prove that the following three identities are all equivalent:

$$\mathbf{P}[A \mid B] = \mathbf{P}[A], \qquad \mathbf{P}[B \mid A] = \mathbf{P}[B], \qquad \mathbf{P}[A, B] = \mathbf{P}[A]\,\mathbf{P}[B].$$

(That is, if A and B satisfies any one of the identities above, then it automatically satisfies the other two.)

Exercise C.56. One can also consider the bisection problem in directed graphs. Here the goal is to find a vertex set S of size $\lfloor n/2 \rfloor \leq |S| \leq \lceil n/2 \rceil$ minimizing the cost of the directed cut $c(\delta^+(S))$. Suppose one had access to a $O(\log n)$ approximation algorithm for uniform directed sparsest cut (as described in **??**). Using this as a subroutine, design and analyze an algorithm that obtains a bicriteria approximation algorithm for the minimum directed bisection problem with essentially the same approximation bicriteria for the undirected setting: compute a set S with $n/3 \leq |S| \leq 2n/3$ with cost $c(\delta^+(S))$ at most a $O(\log n)$-factor greater than that of the minimum directed bisection.

Exercise C.57. The following problem, called the *buy-at-bulk network design* problem, models the situation where you are building out a network that supports communication between various terminal pairs with minimum total cost, with "economies of scale" built into the cost function.

Formally, we have an undirected graph $G = (V, E)$ with edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$. There are k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$, each associated with a demand $d_i > 0$. The high-level goal is to choose a walk $w_i : s_i \rightsquigarrow t_i$ for all i minimizing the "cost" of buying enough capacity of each capacity to route all these paths simultaneously. To model the cost, we are given a monotone subadditive function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, and for each edge, we pay $f(u)$ for capacity u per unit length. *Montone* means that $f(u)$ is nondecreasing in u. *Subadditive* means that $f(u_1 + u_2) \leq f(u_1) + f(u_2)$ for $u, v \geq 0$.

Now, each edge must have enough capacity to route all the demand of all the paths using it. Thus the total cost of a selection of paths $\{p_i : s_i \rightsquigarrow t_i : i = 1, \ldots, k\}$ is

$$\sum_{e \in E} \ell(e) f\left(\sum_{i : e \in w_i} d_i\right).$$

This problem is NP-Hard, so instead we will try to approximate it. Our goal is to design and analyze a polynomial time approximation algorithm for the buy-at-bulk network design problem.

We will use the randomized tree metric of Section 12.3. The algorithm is as follows. We first compute a randomized tree metric d_T, with underlying weighted tree $(T = (V_T, E_T), \ell_T : E_T \to \mathbb{R}_{\geq 0})$, over the shortest path metric of G with respect to ℓ. Treat T as a graph with edge lengths ℓ_T, and mapping the terminal pairs (s_i, t_i) to the corresponding leaves of T, we solve the buy-at-bulk network design problem over T with the same cost function f. We then map this solution back to walks in G and return this solution.

To elaborate on the second point, recall that every edge node in T is associated with a cluster centered at some vertex in G. Each edge e in T is supported by the shortest path in G between the vertices associated with the endpoints of e. Denoting this path by P_e for $e \in E_T$, we have $\ell(P_e) \leq \ell_T(e)$. In this way, every path P in T maps to a walk W in G obtained by concatenating the walks supporting the (tree) edges in P.

We break down the analysis into the following steps.

1. Design and analyze an algorithm that solves the buy-at-bulk network design problem exactly in a tree. (This shows that the first step is exact.)

2. Describe a mapping from solutions in G to solutions in the randomized tree metric T such that for a fixed solution in G, in expectation (over T), the cost of the mapped solution in T is at most a $O(\log n)$ factor greater than original cost in G.

3. Show that for every solution in T, there is a solution in G where the cost in G is at most the cost in T.

4. Combine the three parts above to prove that the randomized solution returned by our algorithm has expected cost $O(\log n)\,\mathrm{OPT}$.

Exercise C.58. A funny characteristic of the distinct elements estimator in Section 8.1 is that it is not monotone – $|B|$ can go down when the next element e_i kicks out a previously sampled copy of the same element – even though the true number of distinct elements is nondecreasing.

Design and analyze a streaming algorithm with the same performance as the distinct elements algorithm, with the additional property that the estimate is nondecreasing.

Exercise C.59. You have a sequence of n switches $S_1, \dots, S_n$ that jointly control m light bulbs $L_1, \dots, L_m$. Each switch can be "up" or "down", and this controls whether the light bulbs are on or off.

Each light bulb L_i, is associated with two sets of switches $A_i, B_i \subseteq [n]$. The switches in A_i turn on the light bulb when they are "up" and the switches in B_i turn on the light bulb then they are "down".

More precisely, for each $j \in A_i$, having switch S_j "up" automatically turns on the light bulb. (It only takes one of these switches to be "up" to turn on the light bulb.) For each $j \in B_j$, turning the switch "down" automatically turns on the light bulb. (Again, it only takes one of these switches to be "down" to turn on the light bulb.)

Thus, for a light bulb L_i, the light bulb L_i lights up if and only if either (a) some switch in A_i is flipped up or (b) some switch in B_i is flipped down. A_i and B_i are generic subsets of switches, not necessarily disjoint, and their union does not necessarily include all the switches. We do assume, however, that $|A_i| + |B_i| \geq 2$ for all i. We assume that the sets A_i and B_i are given explicitly for each i (for simplicity; otherwise they can be obtained by inspection).

Your algorithm can flip switches "up" and "down". For the sake of running times, assume that flipping a single switch takes $O(1)$ time, and inspecting whether a single light bulb is on or off takes $O(1)$ time. The light bulbs turn on and off instantly when you flip a switch.

For each of the following decision problems, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

1. Decide if there exists a way to flip the switches to turn on all the light bulbs.
2. Decide if there exists a way to flip the switches to turn on at least three-fourths of the light bulbs.

Exercise C.60. Using only fact 9.3, show that for $x \sim \mathcal{N}(\mu, \sigma^2)$ and $\alpha \in \mathbb{R}$,

$$\alpha x \sim \mathcal{N}\Big(\alpha\mu, \alpha^2\sigma^2\Big).$$

Exercise C.61. Prove that the multilinear extension $F_{\text{ML}}(x)$ is multilinear function. (This means that for all fixed x, and elements $e \in \mathcal{N}$, $f(x + \delta 1_e)$ is a linear function in δ over all δ such that $x_e + \delta \in [0, 1]$.)

Exercise C.62. Last week you decided to clean up your toolshed, but you compulsively made the following silly mistake. Originally you had n matching pairs of nuts and bolts of different sizes, but you decided to organize the nuts and bolts separately into a box full of nuts and a box full of bolts, splitting up all the pairs of nuts and bolts in the process. So now you have n nuts, and n bolts, such that each nut fits a distinct bolt and vice-versa, but you don't know which nut goes with which bolt.

The bolts all look pretty similar, so you can't compare two bolts directly with each other and tell which one is bigger. Likewise you cannot compare the nuts to each other. However, you can try to fit one nut to one bolt, and see if either:

1. The nut fits the bolt.
2. The nut is too big for the bolt.
3. The nut is too small for the bolt.

We will treat one of these nut-to-bolt tests as a single operation. Your goal is to match up all nuts and bolts. Of course you can compare every pair of nut and bolt in $O(n^2)$ time, but can you do better?

Design and analyze an algorithm, as fast as possible, to recover all matching pairs of nuts and bolts.

Exercise C.63. The goal of this exercise is to show how to get constant time access for n keys with $O(n)$ space, using only universal hash functions. We will require the following fact that we ask you to prove.

1. Let $h : [n] \to [k]$ be a *universal* hash function. Show that for $k \geq n^2$, h has no collisions with probability $\geq 1/2$.

Now we describe the data structure. We first allocate an array $A[1..n]$ of size n. We have one universal hash function h_0 into $[n]$. If we have a set of (say) k collisions at an array cell $A[i]$, rather than making a linked list of length k, and we build another hash table, with a new universal hash function h_i, of size k^2, with no collisions (per part 1). (We may have to retry if there is a collision.) If the total size (summing the lengths of the first arrray and each of the second arrays) comes out to bigger than (say) $5n$, we try again.

2. For each $i = 1, \ldots, n$, let k_i be the number of keys that hash to the ith cell. We have

$$\text{(sum of array sizes of our data structure)} \leq n + \sum_{i=1}^{n} k_i^2.$$

Show that[25]

$$\sum_{i=1}^{n} k_i^2 \leq n + O(\text{total \# of collisions (w/r/t } h_0\text{)}).$$

[25]Here a "collision" is an unordered pair of keys with the same hash. The $O(\cdots)$ means you can choose whatever constant you find convenient; 2 is possible.

3. Show that

$$\mathbf{E}[\text{total \# of collisions (w/r/t } h_0)] \leq n/2.$$

4. Show that

$$\mathbf{P}[(\text{sum of all array sizes}) > Cn] \leq 1/2$$

for some constant $C > 0$. ($C = 5$ is possible.)

Taken together, steps 1 to 3 above show that this approach will build a "perfect" hash table over the n keys in $O(n)$ space with probability of success at least $1/2$, using only universal hash functions. Even if it fails to work, we can then keep repeating the construction until it succeeds. This approach works better in *static settings*, when the set of keys is fixed.

Exercise C.64. Design and analyze a deterministic 3/4-approximation algorithm for max-SAT.[26]

Exercise C.65. Recall that we can obtain a $O(\log m)$-approximation for set cover by randomly rounding the LP. There are also natural special cases where the points and sets have geometric structure; here we consider one such setting.

Suppose we have a set of m disks $D = \{d_1, \ldots, d_m \subseteq \mathbb{R}^2\}$, and n points $P = \{p_1, \ldots, p_n \in \mathbb{R}^2\}$. We say that a point p_j *hits* a disk D_i if $p_j \in D_i$. (You can assume you can query if p_j hits D_i in $O(1)$ time.) A set of points $Q \subseteq P$ is a *hitting set* of D if every disk is hit by some point in Q.

Consider the problem of computing the minimum cardinality hitting set Q of D. This is a special case of set cover, where points "cover" the disks that they hit. Design and analyze a (polynomial-time) approximation algorithm for this problem. Here, the smaller the approximation ratio, the better, but we will not emphasize constant factors. (Yes, you can do better than $O(\log m)$.)

Exercise C.66. Suppose you are going to the graduate student social thingy on the 3rd floor of Lawson.[27] You can get to the third floor by either an elevator or the stairs. The elevator takes 15 seconds (once you get in), while the stairs take 2 minutes. Your goal is to get up to the third floor as fast as possible before the donuts are all taken.

You press the button to go up for the elevator. You don't know how long it will take to come down. Do you wait or take the stairs?

[26]You may first want to design and analyze a deterministic $(1 - 1/e)$-approximation algorithm for max-SAT.

[27](It used to be on the third floor.)

1. Suppose you knew how long the elevator would take to arrive. What is the optimal choice, based on this wait?
2. Now suppose you don't know how long the elevator would take. Design a deterministic algorithm that is (15/8)-competitive with the optimal solution (where you know how long the elevator would take).
3. Suppose you have a quarter in your pocket, which lands heads or tails with equal probability. You can toss the coin once every 15 seconds. Design a randomized algorithm with a (slightly) better competitive ratio than the (best) deterministic one from the previous question.[28]

Exercise C.67. For the n-vertex cycle C_n, describe a randomized tree metric where each edge has expected stretch $O(1)$.

Exercise C.68. Let X, Y be two independent and identically distributed real random variables. Prove that $\mathbf{P}[|X - Y| \leq 2] \leq 3\,\mathbf{P}[||X - Y| \leq 1|]$.

Exercise C.69. Prove Lemma 11.4.

Exercise C.70. Let $G = (V, E)$ be an undirected graph with m edges and n vertices, vertex weights $w : V \to \mathbb{R}_{>0}$, and maximum (unweighted) vertex degree Δ. We let $W = \sum_{v \in V} w(v)$ denote the total weight of all the vertices.

Recall than an *independent set* is a set of vertices $S \subseteq V$ such that no two vertices in S are connected by an edge. Computing the maximum cardinality independent set is NP-Hard. In fact, for all $\epsilon > 0$, it is NP-Hard to get a $1/n^{1-\epsilon}$-approximation to the maximum cardinality independent set. (Of course, computing the maximum *weight* independent set is no easier.)

Consider the following algorithm that always returns a *maximal* independent set:

```
randomized-greedy($G = (V, E)$, $w$)
1. Order the vertices $v_1, \dots, v_n$ uniformly at random.
2. $S \leftarrow \emptyset$.
3. For $i = 1, \dots, n$:
    A. If $S + v_i$ is independent, then set $S \leftarrow S + v_i$.
4. Return $S$.
```

[28] I actually don't know what the best competitive ratio would be, and I'm interested to see what everyone comes up with.

1. (6 points) Analyze the approximation ratio of `randomized-greedy`, as a function of the maximum degree Δ.
2. (4 points) Derandomize `randomized-greedy`. That is, design and analyze a deterministic polynomial-time algorithm that outputs an independent set with weight at least as good as the expected weight of the independent set returned by `randomized-greedy`.

Exercise C.71. Prove that the $O(\log n)$ bound is tight for tree metrics (up to constants).

Appendix D

CS588: Syllabus, Policies, and Procedures

Welcome to CS588, which is about randomized algorithms. Please see the schedule (Page 1) for a tentative list of topics. The class will be similar to the Fall 2024 course (cf. https://fundamentalalgorithms.com/randomized/f24).

Lectures are delivered on

Tuesday and Thursday, from 4:30 to 5:45 PM, in Forney Hall, Room G124.

Please take advantage of class time and office hours to ask questions or express any concerns. Please reserve email only for true emergencies, which I do not expect to arise.

The lectures are accompanied by lecture notes (usually one chapter per lecture) and you are expected to be informed of their contents. I am (tentatively) planning to record the lectures and put them online (say, by the end of the week), as well as upload handwritten slides from the lecture. Links to these resources are provided at the end of each lecture. I caution that the recordings are meant to supplement in-class lectures and should not be regarded as a substitute.

If you are unable to register for the class because it is full, just wait. Historically, slots have always opened up. You can attend class, add yourself to gradescope and submit homework in the meantime.

D.1 Textbooks

No textbook is strictly required as lecture notes[1] are provided. That said, the most closely aligned textbook with our course is:

- Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995

Other books and monographs overlapping with the course include:

- Noga Alon and Joel H. Spencer. *The Probabilistic Method.* 4th. Wiley Publishing, 2016

[1]The notes were first written in Fall 2020 to compensate for remote learning.

- Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. URL: https://doi.org/10.1017/CBO9780511813603
- Salil P. Vadhan. "Pseudorandomness". In: *Foundations and Trends in Theoretical Computer Science* 7.1-3 (2012), pp. 1–336. URL: https://people.seas.harvard.edu/~salil/pseudorandomness/pseudorandomness-published-Dec12.pdf
- Jelani Nelson. "Sketching Algorithms". Lecture notes. Dec. 2020. URL: https://www.sketchingbigdata.org/fall20/lec/notes.pdf
- David P. Woodruff. "Sketching as a Tool for Numerical Linear Algebra". In: *Found. Trends Theor. Comput. Sci.* 10.1-2 (2014), pp. 1–157. URL: https://doi.org/10.1561/0400000060
- Daniel A. Spielman. "Spectral and Algebraic Graph Theory". Draft. 2019. URL: https://www.cs.yale.edu/homes/spielman/sagt/sagt.pdf
- Luca Trevisan. *Lecture Notes on Graph Partitioning, Expanders, and Spectral Methods*. Spring 2016. URL: https://lucatrevisan.github.io/books/expanders-2016.pdf
- Sariel Har-Peled. *Geometric Approximation Algorithms*. USA: American Mathematical Society, 2011

The following courses from other institutions also have lecture notes that you may find helpful.

- *Randomized algorithms*, taught by David Karger at MIT. http://courses.csail.mit.edu/6.856/current/
- *Randomized algorithms*, taught by Sariel Har-Peled at UIUC. https://sarielhp.org/teach/17/b/
- *Randomized algorithms*, taught by Anupam Gupta and Avrim Blum at CMU. http://www.cs.cmu.edu/~avrim/Randalgs11/index.html
- *Randomized algorithms and probabilistic analysis*, taught by Greg Valiant at Stanford. http://theory.stanford.edu/~valiant/teaching/CS265/index.html
- *Algorithms for Big Data*, taught by Chandra Chekuri at UIUC. https://courses.engr.illinois.edu/cs498abd/fa2020/.

D.2 Correspondence

The course website is

www.fundamentalalgorithms.com/randomized,

where this document is posted.

Email. Please reserve emails to the instructor for true emergencies. Please use class time or office hours to ask questions or express concerns.

Piazza. There is a Piazza for the course at the following address.

https://piazza.com/purdue/fall2025/cs588

The point of Piazza is to foster discussion among the students. The TA will monitor Piazza regularly (but not continuously) on weekdays during reasonable hours.

D.3 Grading

- 30% Homework
- 30% Midterm
- 40% Final

We compute numerical scores based on the weighting above (as a fractional value between 0 and 1), and then we curve the grades.

D.4 Exams

The midterms and final are each one part multiple choice and one part word problems. We have reserved classrooms through the school to give students an extended period of time for the tests.

D.4.1 What's on the midterm?

I have not made the test yet, so I cannot speak with much certainty. That said, two broad goals are:

1. Help identify weak points in each student's understanding of the material, so we can address them.
2. (Gently) incentivize the class to learn all the material, stay engaged, etc.

So I plan to try to cover everything we've covered, including the material the week before the exam (though perhaps slightly less emphasized). I expect to have a mix of multiple-choice or short-answer problems and a few longer word problems.

I will also prepare some kind of cheat sheet with all the inequalities we use, such as Markov's inequality, Chernoff's inequality, etc. I will send everyone this cheat sheet before the exam.

Here are some ways to practice and study:

- Reviewing the lectures (obviously).

- Making sure you understand the solutions to the homework, even if you didn't get them right the first time.
- There are many more problems in the notes that I did not assign. While we do not provide solutions for them, I can quickly tell you in person if you are on the right track for any of them. I also encourage students to discuss solutions with one another; perhaps over Piazza.
- The notes include essentially all problems that I have given in past exams.
- I may have problems like "Prove Chebyshev's inequality" or "Prove the multiplicative Chernoff bound (for any constant in the exponent)." So I would learn the proofs of some of these basic tools.
- For some of the recent lectures, it may be helpful to at least attempt the problems in the homework, even if they are not due until after the midterm. The problems are meant to help you understand the material.
- More problems can be found in the Motwani-Raghavan book. Even if we often covered different algorithms, you can still find exercises about more general themes; e.g., conditional probability, linearity of expectation, concentration bounds, etc.
- I found a practice final from Sariel's website here. https://sarielhp.org/teach/17/b/sp14/sp14_final.pdf

D.5 Homework

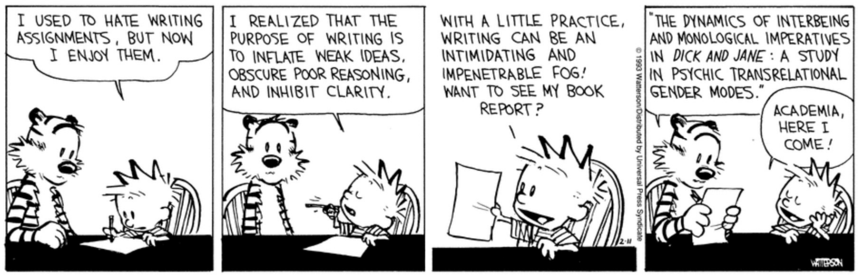
This course has regular homework, generally due every two weeks (with the exception of homeworks 0 and 1.) Homeworks will be due at 11:59PM on Thursday nights.

Typesetting. Homework submissions that are not typeset in LaTeX or equivalent will not be graded. Some tips on typesetting are listed below. A simple Overleaf template is set up at https://www.overleaf.com/read/fczzqbftywcp.

On writing. The onus is on the student to make the arguments in their solution clear, and points will be docked if the grader cannot easily verify that the solution is correct. The class is as much about *communicating complicated ideas* as solving problems and applying techniques. Particularly clear exposition may be selected as homework solutions which is rewarded with extra credit (see below).

Here are some articles about writing:

- Terry Tao: https://terrytao.wordpress.com/advice-on-writing-papers/.
- Cormac McCarthy: https://www.nature.com/articles/d41586-019-02918-5. (Accessible via school library.)


I USED TO HATE WRITING ASSIGNMENTS, BUT NOW I ENJOY THEM.
I REALIZED THAT THE PURPOSE OF WRITING IS TO INFLATE WEAK IDEAS, OBSCURE POOR REASONING, AND INHIBIT CLARITY.
WITH A LITTLE PRACTICE, WRITING CAN BE AN INTIMIDATING AND IMPENETRABLE FOG! WANT TO SEE MY BOOK REPORT?
"THE DYNAMICS OF INTERBEING AND MONOLOGICAL IMPERATIVES IN DICK AND JANE: A STUDY IN PSYCHIC TRANSRELATIONAL GENDER MODES."
ACADEMIA, HERE I COME!
WATTERSON

Gradescope (subject to change). The word problems will be collected online at gradescope.com. The multiple choice questions will be posted on gradescope.com. as well. If you are registered for the course on BrightSpace, then you should have been automatically added to gradescope. Otherwise you can add yourself with the code 7X7WXK.

Collaboration. Collaboration is allowed and interaction among students is encouraged. Currently we are allowing up to three students per submission. Please also indicate any other students (outside your group) that you may have worked on the problems with.

Dropping scores. In the overall homework grade, the bottom one-fifth of word problem scores will be dropped. More precisely, if there are n total world problems assigned in homework, then the $\lceil n/5 \rceil$ lowest scores will be dropped. This is largely to help cover the arbitrary exceptions that arise throughout a semester.

Late policy (subject to change). For word problems, we have a simple late policy where you can submit up to *12 hours* late, at a cost of 10% of the total points.

There are no exceptions to the late policy. We expect the $\lceil n/5 \rceil$-dropped scores to absorb most scenarios that arise; besides, 35% off is not the end of the world.

In rare circumstances accompanied by documentation by the Dean of Students we may instead give a 0/0 for all problems on that assignment.

We plan to put up the solutions quickly after the homework is collected.

Selected solutions. The staff will select exemplary submissions and publish them as solutions. If you have a strong preference to be excluded from consideration for a particular homework problem, please indicate it clearly and explicitly at the top of your submission (for each problem). For this reason, *please leave your student ID off of your submission.* If you have a strong preference to be anonymous if your homework is selected, please indicate that on your document.

Selected solutions will get 10% extra credit.

Self-grading In part because of a very high ratio of students to staff, we are experimenting with a new (pseudo-) self-grading system. The idea is as follows:

1. After you submit the homework on gradescope, we will post solutions and rubrics (in Chapter B or on Piazza).

2. A week after the first submission deadline, you will grade your own submission based on the posted rubric, and upload the same submission (we will check!) with your self-grading appended at the end. This includes:
   (a) Numerical score broken down according to the rubric.
   (b) Additional comments as needed (e.g., explaining why you gave yourself half-credit, high-level description on what went awry, etc.)

Tanmay and Zilin will then use the self-grading as a guide when he actually grades your submission. We expect you to be honest in your self-grading and we will be annoyed and disappointed if you aren't.

On a case by case basis, the rubrics for a problem may be based on a particular solution differing from your approach, making it harder to apply the rubric. In this case we ask you to try your best and be reasonable interpreting and adjusting the rubric to your case. (Maybe some points, like stating the running time, still carry over. Maybe you can still get a sense of how many of the points are for the implementation, and how many for the analysis, and judge appropriately.) The key point is to save the grader's time, while also nudging you to critically evaluate your own submission and possibly learn from the posted solutions.

IDK. One may simply write "I don't know" or "IDK" and automatically get 25% of the possible points (for any problem or subproblem).

MOM SAID I CAN'T GO OUTSIDE UNTIL I FINISH MY HOMEWORK. IF YOU'LL HELP ME, I'LL BE DONE FASTER. WHAT'S FIVE PLUS SEVEN?
I DON'T KNOW.
I DON'T EITHER.
THEN WRITE, "I DON'T KNOW."
HEY, THAT'S A TRUE ANSWER, ISN'T IT! I CAN WRITE THAT FOR ALL OF THESE! WE'RE DONE!
WE'D BETTER HAVE A LOOK AT OUR PRODIGY'S HOMEWORK.
WATTERSON

Regrades. Regrade request must be initiated within one week of the grades being returned.

Typesetting tips.

- The standard for typesetting mathematical and scientific articles is LaTeX. Even if you do not know LaTeX now, you probably have to learn it sooner or later (and certainly if you pursue graduate studies).

- The instructor uses `emacs` to write LaTeX, but any editor will do. There is also a website called `overleaf.com` for typesetting LaTeX.
- Alternatively, the open-source software `marktext` allows one to write LaTeX within a markdown document, which is particularly easy to use.
- `LyX` is another popular latex editor that is WYSIWYG.
- There are several apps for scanning documents (e.g., when inserting pictures) that are much better than taking a photo. The instructor uses `scanbot`, and other popular apps include `microsoft office lens`, `camscanner`, and `evernote scannable`.

D.6 Academic integrity

Behavior consistent with cheating, copying, and academic dishonesty is not tolerated. Depending on the severity, this may result in a zero score on the assignment or exam, and could result in a failing grade for the class or even expulsion. Purdue prohibits "dishonesty in connection with any University activity. Cheating, plagiarism, or knowingly furnishing false information to the University are examples of dishonesty." (Part 5, Section III-B-2-a, University Regulations) Furthermore, the University Senate has stipulated that "the commitment of acts of cheating, lying, and deceit in any of their diverse forms (such as the use of substitutes for taking examinations, the use of illegal cribs, plagiarism, and copying during examinations) is dishonest and must not be tolerated. Moreover, knowingly to aid and abet, directly or indirectly, other parties in committing dishonest acts is in itself dishonest." (University Senate Document 7218, December 15, 1972). You are expected to read both Purdue's guide to academic integrity (`http://www.purdue.edu/purdue/about/integrity_statement.html`) and Prof. Gene's Spafford's guide (`http://spaf.cerias.purdue.edu/integrity.html`) as well. You are responsible for understanding their contents and how it applies to this class.

D.7 Posting Class Material

Posting material associated with this class (e.g., solutions to homework sets or exams) without the written permission of the instructor is forbidden and may be a violation of copyright.

D.8 Purdue's Honor Pledge

As a boilermaker pursuing academic excellence, I pledge to be honest and true in all that I do. Accountable together - we are Purdue. `https://www.purdue.edu/provost/teachinglearning/honor-pledge.html`.

D.9 Grief Absence Policy

Purdue University recognizes that a time of bereavement is very difficult for a student. The University therefore provides the following rights to students facing the loss of a family member through the Grief Absence Policy for Students (GAPS). According to GAPS Policy, students will be excused for funeral leave and given the opportunity to earn equivalent credit and to demonstrate evidence of meeting the learning outcomes for missed assignments or assessments in the event of the death of a member of the student's family.

D.10 Conduct and Courtesy

Students are expected to maintain a professional and respectful classroom environment. This includes: silencing cellular phones, arriving on time for class, speaking respectfully to others and participating in class discussion. You may use non-disruptive personal electronics for the purpose class participation (e.g., taking notes).

D.11 Students with Disabilities

Purdue University is required to respond to the needs of the students with disabilities as outlined in both the Rehabilitation Act of 1973 and the Americans with Disabilities Act of 1990 through the provision of auxiliary aids and services that allow a student with a disability to fully access and participate in the programs, services, and activities at Purdue University. If you have a disability that requires special academic accommodation, please make an appointment to speak with the instructor within the first three (3) weeks of the semester in order to discuss any adjustments.

It is the student's responsibility to notify the Disability Resource Center (`http://www.purdue.edu/drc`) of an impairment/condition that may require accommodations and/or classroom modifications. We cannot arrange special accommodations without confirmation from the Disability Resource Center.

D.12 Emergencies

In the event of a major campus emergency, course requirements, deadlines and grading percentages are subject to changes that may be necessitated by a revised semester calendar or other circumstances beyond the instructor's control. Relevant changes to this course will be posted onto the course website and/or announced via email. You are expected to read your purdue.edu email on a frequent basis. Emergency Preparedness: Emergency notification procedures are based on a simple concept: If you hear an alarm inside, proceed outside. If you hear a siren outside, proceed inside. Indoor Fire Alarms are mean to stop class or research and immediately evacuate the building. Proceed to your Emergency

Assembly Area away from building doors. Remain outside until police, fire, or other emergency response personnel provide additional guidance or tell you it is safe to leave. All Hazards Outdoor Emergency Warning sirens mean to immediately seek shelter (Shelter in Place) in a safe location within the closest building. "Shelter in place" means seeking immediate shelter inside a building or University residence. This course of action may need to be taken during a tornado, a civil disturbance including a shooting or release of hazardous materials in the outside air. Once safely inside, find out more details about the emergency. Remain in place until police, fire, or other emergency response personnel provide additional guidance or tell you it is safe to leave. In both cases, you should seek additional clarifying information by all means possible: Purdue Home page, email alert, TV, radio, etc. Review the Purdue Emergency Warning Notification System multi-communication layers at http://www.purdue.edu/ehps/emergencypreparedness/warning-system.html. Please review the Emergency Response Procedures at https://www.purdue.edu/emergencypreparedness/flipchart/index.html. Please review the evacuation routes, exit points, emergency assembly area and shelter in place procedures and locations for the building. Video resources include a 20-minute active shooter awareness video that illustrates what to look for and how to prepare and react to this type of incident. See http://www.purdue.edu/securepurdue/police/video/

D.13 Violent Behavior Policy

Purdue University is committed to providing a safe and secure campus environment for members of the university community. Purdue strives to create an educational environment for students and a work environment for employees that promote educational and career goals. Violent Behavior impedes such goals. Therefore, Violent Behavior is prohibited in or on any University Facility or while participating in any university activity.

D.14 Mental Health and Wellness

If you find yourself beginning to feel some stress, anxiety and/or feeling slightly overwhelmed, try Therapy Assistance Online (TAO) (https://www.purdue.edu/caps/students/resources/self-help/digital/tao.php), a new web and app-based mental health resource available courtesy of Purdue Counseling and Psychological Services (CAPS). TAO is available to students, faculty, and staff at any time.

If you need support and information about options and resources, please contact or see the Office of the Dean of Students (www.purdue.edu/odos). Call 765-494-1747. Hours of operation are M-F, 8 am- 5 pm.

If you find yourself struggling to find a healthy balance between academics, social life, stress, etc. sign up for free one-on-one virtual or in-person sessions with a

Purdue Wellness Coach at RecWell (`https://www.purdue.edu/recwell/fitness-wellness/wellness/one-on-one-coaching/wellness-coaching.php`). Student coaches can help you navigate through barriers and challenges toward your goals throughout the semester. Sign up is completely free and can be done on BoilerConnect. If you have any questions, please contact Purdue Wellness at `evans240@purdue.edu`.

If you're struggling and need mental health services: Purdue University is committed to advancing the mental health and well-being of its students. If you or someone you know is feeling overwhelmed, depressed, and/or in need of mental health support, services are available. For help, such individuals should contact Counseling and Psychological Services (CAPS) (`https://www.purdue.edu/caps/`) at 765-494-6995 during and after hours, on weekends and holidays, or by going to the CAPS office on the second floor of the Purdue University Student Health Center (PUSH) during business hours.

Purdue University is committed to advancing the mental health and well-being of its students. If you or someone you know is feeling overwhelmed, depressed, and/or in need of support, services are available. For help, such individuals should contact Counseling and Psychological Services (CAPS) at (765) 494-6995 and `http://www.purdue.edu/caps/` during and after hours, on weekends and holidays, or through its counselors physically located in the Purdue University Student Health Center (PUSH) during business hours.

D.15 Health in general

In general, if medical conditions prohibit you from participating in the class, please be proactive in seeking professional medical care. The link to the Purdue University Student Health Center (PUSH) is listed below:

> `https://www.purdue.edu/push/`.

In cases falling under excused absence regulations, the student or the student's representative should contact or go to the Office of the Dean of Students (ODOS, `https://www.purdue.edu/advocacy/students/absence-policies.html`) website to complete appropriate forms for instructor notification. Under academic regulations, excused absences may be granted by ODOS for cases of grief/bereavement, military service, jury duty, parenting leave, or emergent or urgent care medical care.

No one on the teaching staff is qualified to make any kind of diagnosis, and we rely the dean of students (who are suppose to be able to handle medical situations) to document serious medical cases and provide us with instructions when applicable. We have recourse policies in place for documented illness.

D.16 Basic needs and security

Any student who faces challenges securing their food or housing and believes this may affect their performance in the course is urged to contact the Dean of Students for support. There is no appointment needed and Student Support Services is available to serve students 8 a.m.-5 p.m. Monday through Friday. Considering the significant disruptions caused by the current global crisis as it relates to COVID-19, students may submit requests for emergency assistance from the Critical Need Fund (`https://www.purdue.edu/odos/resources/critical-need-fund.html`).

D.17 Nondiscrimination

Purdue University is committed to maintaining a community which recognizes and values the inherent worth and dignity of every person; fosters tolerance, sensitivity, understanding, and mutual respect among its members; and encourages each individual to strive to reach his or her own potential. In pursuit of its goal of academic excellence, the University seeks to develop and nurture diversity. The University believes that diversity among its many members strengthens the institution, stimulates creativity, promotes the exchange of ideas, and enriches campus life. Purdue University prohibits discrimination against any member of the University community on the basis of race, religion, color, sex, age, national origin or ancestry, marital status, parental status, sexual orientation, disability, or status as a veteran. The University will conduct its programs, services and activities consistent with applicable federal, state and local laws, regulations and orders and in conformance with the procedures and limitations as set forth in Executive Memorandum No. D-1, which provides specific contractual rights and remedies.

D.18 Privacy

The Federal Educational Records Privacy Act (FERPA) protects information about students, such as grades. If you apply for a job and wish to use the instructor as a reference, you should tell the instructor beforehand. Otherwise, the instructor cannot say anything about you to a prospective employer who might call. The instructor is happy to provide references and to write letters of recommendation for his students as needed.

D.19 Netiquette

We want to foster a safe online learning environment. All opinions and experiences, no matter how different or controversial they may be perceived, must be respected in the tolerant spirit of academic discourse. You are encouraged to comment, question, or critique an idea, but you may not attack an individual. Our differences, some of which are outlined in the

University's nondiscrimination statement below, will add richness to this learning experience. Please consider that sarcasm and humor can be misconstrued in online interactions and generate unintended disruptions. Working as a community of learners, we can build a polite and respectful course ambience. Please read the Netiquette rules for this course:

- Monitor how much space/time you are taking up in any discussion. Give other students the opportunity to join in the discussion.
- Do not use offensive language. Present ideas appropriately.
- Be cautious in using Internet language. For example, do not capitalize all letters since this suggests shouting.
- Avoid using vernacular and/or slang language. This could lead to misinterpretation.
- Keep an "open-mind" and be willing to express even your minority opinion.
- Think and edit before you push the "Send" button.
- Seek and take in feedback from others; learning from other people is an important life skill.

D.20 Changes to the syllabus

This syllabus is subject to change and changes will be announced appropriately.