Randomized Algorithms, CS588, Fall 2022

Lectures: 4:30 to 5:45 PM, on Tuesdays and Thursdays, in Lawson room 1106.

Staff: Email (@purdue.edu) Office hours
Kent Quanrud       krq       After class, LWSN 1211
Jawad “JD” Raheel (TA) jraheel     Wed., 10–11AM, Haas G50

Please see the syllabus (page 315) for more information.

Homework

- Homework 0 due August 31.
- Homework 1 due September 16.
- Homework 2 due September 28.

Class schedule\(^1\)

1. August 23. Randomized SAT and quicksort (chapter 1).
3. August 30. Hash tables and linear probing (chapter 3).
4. September 1. Randomized minimum cut (sections 4.1–4.3).
8. September 15. Dimensionality reduction (chapter 8).

\(^1\)All future dates should be regarded as tentative.
11. September 27. Sampling geometric range spaces, and PAC learning (chapter 10).

(⋆) Midterm on Thursday, October 13, at 8PM in Hampton Hall, room 1252.
15. October 18. Lovász local lemma.
18. October 27. Discrepancy via high-dimensional random walks.

(⋆) Thanksgiving break.
27. December 6. Randomly testing boolean functions (chapter 21).
28. December 8. Last day of class.
# Contents

## Basic information 1

## Class schedule 1

### 1 Randomized searching and sorting 9

1.1 First randomized algorithms .................................. 9
1.2 Basic probability .............................................. 13
1.3 Randomized approximations for SAT .......................... 19
1.4 Randomized sorting .......................................... 23
1.5 Additional notes and materials ................................. 27
1.6 Exercises ...................................................... 27

### 2 Hashing and heavy hitters 33

2.1 Introduction .................................................. 33
2.2 Hashing ....................................................... 36
2.3 Using hashing to approximate frequencies ................... 37
2.4 Amplification .................................................. 39
2.5 Extensions .................................................... 40
2.6 Takeaways ..................................................... 41
2.7 Additional notes and references ............................... 42
2.8 Exercises ..................................................... 43

### 3 Hash tables and linear probing 47

3.1 Dictionaries ................................................... 47
3.2 Hash tables with chaining ..................................... 51
3.3 Linear probing ................................................ 52
3.4 4-wise independence .......................................... 57
3.5 Takeaways ..................................................... 60
3.6 Additional notes and materials ............................... 60
3.7 Exercises ..................................................... 61
# Contents

4 Sampling edges  63
  4.1 Minimum cut  ........................................ 63
  4.2 Amplification by branching  ..................... 68
  4.3 Randomized branching  ............................ 71
  4.4 Sparsification  .................................. 72
  4.5 Randomized Ford-Fulkerson  ................... 81
  4.6 Additional notes and materials  .......... 87
  4.7 Exercises  .................................... 87

5 Random Sums and Graphs  90
  5.1 Random sums  .................................... 90
  5.2 Random graphs  ................................ 92
  5.3 A gap in component size  ..................... 96
  5.4 Galton-Watson branching processes for general branch factor  . 99
  5.5 Exercises  .................................... 100
  5.6 Additional notes and materials  .......... 101

6 Randomized Rounding  102
  6.1 SAT  ........................................... 104
  6.2 Set cover  ...................................... 108
  6.3 Covering integer programs  .............. 110
  6.4 Exercises  .................................... 113

7 Distinct elements  115
  7.1 Unique visitors  ................................ 115
  7.2 Preliminaries on continuous random variables  ............ 117
  7.3 Distinct elements, continued  .............. 118
  7.4 Variance and Chebyshev’s inequality  ....... 121
  7.5 Amplification: variance reduction and the median trick  .... 121
  7.6 Distinct elements with pairwise independent hash functions  . 125
  7.7 Takeaways  ..................................... 125
  7.8 Additional notes and materials  .......... 127
  7.9 Exercises  .................................... 127

8 Dimensionality Reduction  129
  8.1 Large data sets and long vectors  .......... 129
  8.2 Gaussian random variables: an interface  .... 131
  8.3 Random Projections  ............................ 134
  8.4 Gaussians  .................................... 135
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>Additional notes and materials</td>
<td>138</td>
</tr>
<tr>
<td>8.6</td>
<td>Exercises</td>
<td>138</td>
</tr>
<tr>
<td>9</td>
<td>Locality sensitive hashing and approximate nearest neighbors</td>
<td>140</td>
</tr>
<tr>
<td>9.1</td>
<td>Nearest neighbor search</td>
<td>140</td>
</tr>
<tr>
<td>9.2</td>
<td>Locality Sensitive Hashing for Angles</td>
<td>141</td>
</tr>
<tr>
<td>9.3</td>
<td>Locality Sensitive Hashing for Euclidean Distance</td>
<td>143</td>
</tr>
<tr>
<td>9.4</td>
<td>Exercises</td>
<td>148</td>
</tr>
<tr>
<td>10</td>
<td>Sampling geometric range spaces</td>
<td>149</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>149</td>
</tr>
<tr>
<td>10.2</td>
<td>Proof of the $\epsilon$-sample theorem</td>
<td>151</td>
</tr>
<tr>
<td>10.3</td>
<td>Additional notes and materials</td>
<td>152</td>
</tr>
<tr>
<td>11</td>
<td>Sparsest cut</td>
<td>153</td>
</tr>
<tr>
<td>11.1</td>
<td>The LP</td>
<td>154</td>
</tr>
<tr>
<td>11.2</td>
<td>Rounding via region growing (for uniform demands)</td>
<td>155</td>
</tr>
<tr>
<td>11.3</td>
<td>Rounding via $L_1$-metric embeddings</td>
<td>159</td>
</tr>
<tr>
<td>11.4</td>
<td>Expander lower bounds</td>
<td>167</td>
</tr>
<tr>
<td>11.5</td>
<td>Application: Minimum bisection</td>
<td>170</td>
</tr>
<tr>
<td>11.6</td>
<td>Exercises</td>
<td>172</td>
</tr>
<tr>
<td>11.7</td>
<td>Additional notes and references</td>
<td>173</td>
</tr>
<tr>
<td>12</td>
<td>Tree Metrics</td>
<td>174</td>
</tr>
<tr>
<td>12.1</td>
<td>Introduction</td>
<td>174</td>
</tr>
<tr>
<td>12.2</td>
<td>Low-Stretch Spanning Trees</td>
<td>175</td>
</tr>
<tr>
<td>12.3</td>
<td>Hierarchical Tree Metrics</td>
<td>178</td>
</tr>
<tr>
<td>12.4</td>
<td>Exercises</td>
<td>181</td>
</tr>
<tr>
<td>13</td>
<td>Two Theorems by Claude Shannon</td>
<td>182</td>
</tr>
<tr>
<td>13.1</td>
<td>Introduction</td>
<td>182</td>
</tr>
<tr>
<td>13.2</td>
<td>Circuit lower bounds</td>
<td>182</td>
</tr>
<tr>
<td>13.3</td>
<td>Coding theory</td>
<td>185</td>
</tr>
<tr>
<td>13.4</td>
<td>Entropy</td>
<td>190</td>
</tr>
<tr>
<td>13.5</td>
<td>Principle of Independence</td>
<td>192</td>
</tr>
<tr>
<td>13.6</td>
<td>Exercises</td>
<td>194</td>
</tr>
</tbody>
</table>
C.8 Purdue’s Honor Pledge .................................................. 320
C.9 Grief Absence Policy .................................................. 320
C.10 Conduct and Courtesy ............................................... 320
C.11 Students with Disabilities ........................................... 320
C.12 Emergencies ............................................................. 320
C.13 Violent Behavior Policy ............................................... 321
C.14 Mental Health and Wellness ....................................... 321
C.15 Health in general ....................................................... 322
C.16 Basic needs and security ............................................ 323
C.17 Nondiscrimination ..................................................... 323
C.18 Privacy ................................................................. 323
C.19 Changes to the syllabus .............................................. 323
Chapter 1

Randomized searching and sorting

1.1 First randomized algorithms

We introduce randomized algorithms by studying the following 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 all very simple to describe and code. They might seem tricky to analyze, as the analysis involves probabilities and sorting through nondeterministic outcomes. But we will see that by focusing on average performance metrics, they can be analyzed quite cleanly as well.

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.

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,

\[ f(x_1, x_2, x_3) = (x_1 \lor x_2 \lor x_3) \land (\bar{x}_1 \lor \bar{x}_2 \lor \bar{x}_3) \land (x_1 \lor \bar{x}_2 \lor x_3) \land (\bar{x}_1 \lor x_2 \lor x_3) \]

is a 3-SAT formula with \( m = 4 \) clauses and \( n = 3 \) variables. It is satisfied by the assignment \( (x_1, x_2, x_3) = (t, f, t) \).\(^1\) 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 \( P = NP \) [cook; levin]. However 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

\[^1\]Here \( t \) denotes “true” and \( f \) denotes “false”.

9
1. Randomized searching and sorting

1.1. First randomized algorithms

random-SAT($f(x_1, \ldots, x_n)$)

1. For each $i \in [n]$, draw $x_i \in \{t, f\}$ independently and uniformly at random.
2. Return $x_1, \ldots, x_n$.

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 $\alpha$-approximation algorithm for SAT is an algorithm that produces an assignment that satisfies at least $\alpha$ OPT clauses. While obtaining an exact algorithm is NP-Hard, for fixed $\alpha < 1$, it is not necessarily NP-Hard to obtain an $\alpha$-approximation algorithm for SAT.

We will analyze the following exceedingly simple randomized algorithm: given a formula $f(x_1, \ldots, x_n)$, for each variable $x_i$, flip a fair coin and assign $x_i = t$ or $x_i = f$ 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.

Surely, an algorithm as simple as the randomized algorithm above could not be very good. 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 $7/8$th of the clauses. Moreover, for all $\epsilon > 0$, a polynomial time approximation algorithm with approximation ratio $(7/8 + \epsilon)$ implies that $P = NP$. 
1. Randomized searching and sorting

1.1. First randomized algorithms

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.


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 in a list in sorted order. The reader like 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. However, the algorithm is randomized, and 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.) Beyond a $O(n \log n)$ average running time, we will also show that the algorithm takes $O(n \log n)$ time with extremely 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.

In summary we will prove the following theorem in this chapter.

**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.

### 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 $k$th smallest element in the list.

The obvious solution is to sort the list, which takes $O(n \log n)$ time. But in fact one can do better: the “median-of-medians” divide-and-conquer algorithm of BFPRT 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 $\ell$. 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 1.12, employing the new tools gained from analyzing randomized SAT and sorting.
Theorem 1.3. \texttt{quick-select}(A[1..n], k) returns the rank \emph{k} element in $O(n)$ time in expectation and with high probability.

1.2 Basic probability

To analyze these algorithms we need to know some basic probability. Let us start with a familiar example.

Suppose we flip a coin, and then press pause as it is flipping in mid-air. Will it land heads or tails? Obviously, \emph{we don’t know yet}. Yet we can state without ambiguity that \emph{half the time it will land heads}, and \emph{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. We are imagining that, if we repeat the experiment many times, we would expect half the coin tosses to come up heads.

This simple example, which we all understand thoroughly, points to a deeper feature of probability: \emph{probability allows us to interpret 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 \emph{the time} the coin will come up heads.

The formal rules of probability are simple and intuitive. (The tricky part is adhering to them.) Likely the reader has had some acquaintance with random events and variables before, but we will still review the basics. Probability theory assumes an uncertain world where \emph{events} occur with fixed (but not necessarily known) numerical probabilities. Each event $A$ has a probability between 0 and 1, denoted

$$P[A] \in [0, 1].$$

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

$$P[A] + P[\bar{A}] = 1. \quad (1.1)$$

\textbf{Joint events.} For any two events $A$ and $B$, one can define the \emph{conjunctive event} that both $A$ and $B$ occur: denoted

\begin{quote}
\text{“}A \land B\text{” or “}A \cap B\text{” or “}A \text{ and } B\text{”}.
\end{quote}

It is common to write $P[A, B]$ as a shorting for $P[A \land B]$.

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

1. $A \land B$: 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 $B$ occurs.

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:

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

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

$$P[A] = P[A, B] + P[A, \bar{B}]$$

If we divide both sides by $P[A]$, we have

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

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

*of the times that event $A$ occurs, $B$ also occurs $P[A, B]/P[A]$ fraction of the time.*

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

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

Likewise we have the conditional probabilities $P[\bar{B} \mid A]$, $P[B \mid \bar{A}]$, $P[\bar{B} \mid \bar{A}]$, and so forth.

We always have

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

(see exercise 1.1). It is generally not true that

$$P[A \mid B] = P[A]$$ (1.2)
for two events $A$ and $B$. The equation above is equivalent to the one with $A$ and $B$ flipped, as well as

$$P[A \land B] = P[A] P[B]$$  \hspace{1cm} (1.3)

Events $A$ and $B$ are said to be independent events in the special case where eqs. (1.2) and (1.3) holds.

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

"$A \lor B$" or "$A \cup B$" or "$A$ or $B$".

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

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

Consequently we have

$$P[A \lor B] = P[A \land B] + P[A \land \bar{B}] + P[\bar{A} \land B].$$

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


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

Dropping the nonnegative term $P[A \land B]$ from the identity above gives rise to the following extremely useful union bound.

**Lemma 1.4 (Union bound).** For any two events $A$ and $B$,

$$P[A \lor B] \leq P[A] + P[B],$$

with equality iff $P[A \land B] = 0$.

**Random variables.** A finite random variable models an unrealized and uncertain object $X$ that takes one of a finite set of values, $\{x_1, \ldots, x_k\}$.\(^2\) For each outcome

\(^2\)One can also define continuous variable (e.g., that take values continuously between 0 and 1), where sums are replaced by variables.
Randomized searching and sorting

1. Randomized searching and sorting

1.2. Basic probability

x_i, “X equals x_i” is an event, with a fixed probability, denoted P[X = x_i]. These probabilities sum to 1:

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

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

\[ P[X = \text{heads}] = P[X = \text{tails}] = \frac{1}{2}. \]

Note that these two probabilities sum to 1.

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

\[ P[X = x_i, Y = y_j]. \]

It is not generally true

\[ P[X = x_i, Y = y_k] = P[X = x_i] P[Y = y_k] \]

In the special case where the 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.

For example, suppose X, Y ∈ \{heads, 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,

\[ P[X = \text{heads}, Y = \text{heads}] = P[X = \text{heads}, Y = \text{tails}] = \frac{1}{4}, \]

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

\[ P[X = \text{heads}, Y = \text{heads}] = P[X = \text{tails}, Y = \text{tails}] = \frac{1}{2}, \]

while

\[ P[X = \text{heads}, Y = \text{tails}] = P[X = \text{tails}, Y = \text{heads}] = 0. \]

Here, X and Y are not independent.
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 1.5.** 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 $\mathbb{E}[X]$, is the weighted sum

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

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

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

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

The proof of linearity of expectation is given as exercise 1.5. 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:

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

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 \(A\), \(E[X \mid A]\) denotes the expected value of \(X\) conditional on \(A\) occurring. Formally we have

\[
E[X \mid A] = \sum_x x P[X = x \mid A].
\]

The reader should verify that

\[
E[X] = E[X \mid A]P[A] + E[X \mid \bar{A}]P[\bar{A}]
\]

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

\[
E_{X,Y}[f(X,Y)] = \sum_{(x,y)} f(x,y)P[X = x, Y = y]. \tag{1.4}
\]

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

\[
E_X[E_Y[f(X,Y) \mid X]].
\]

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

\[
E_X[E_Y[f(X,Y) \mid X]] = \sum_x E_Y[f(X,Y) \mid X = x]P[X = x]
= \sum_x \sum_y f(x,y)P[Y = y \mid X = x]P[X = x]. \tag{1.5}
\]

Since \(P[X = x, Y = y] = P[Y = y \mid X = x]P[X = x]\), the RHSs of eqs. (1.4) and (1.5) are equal, hence

\[
E_{X,Y}[f(X,Y)] = E_X[E_Y[f(X,Y) \mid X]].
\]

(See also exercise 1.6.)
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 $E[X]$. Suppose it is hard to analyze $E[X]$ directly, but for some contextual reason there is a second random variable $Y$ for which the conditional expectation $E[X | Y]$ is better understood. Then $E[X]$ might be computed indirectly via

$$E[X] = E_{X,Y}[X] = E_Y[E[X | Y]].$$

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; $E[X]$ 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 $E[Y] = P[Y = 1]$, $E[X | Y = 1]$, and $E[X | Y = 0]$. Then we can obtain the probability of it raining tomorrow, $E[X]$, via the law of iteration expectations, as


Another nice example appears soon in section 1.4.

### 1.3 Randomized approximations for SAT

Let us test our new tools on the randomized algorithm for 3-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 $x_i \in \{t, f\}$ accordingly. We claim that this algorithm gives a $(7/8)$th approximation (in expectation).

Consider a single clause; e.g., $(x_1 \lor \bar{x}_2 \lor x_3)$. Of all 8 ways to assign the three variables ($x_1, x_2, x_3$ in the example) values in $\{t, f\}$, 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, for the probability that they are both satisfied, we have

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

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

<table>
<thead>
<tr>
<th></th>
<th>$E_2$</th>
<th>$\bar{E}_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$</td>
<td>49/64</td>
<td>7/64</td>
</tr>
<tr>
<td>$\bar{E}_1$</td>
<td>7/64</td>
<td>1/64</td>
</tr>
</tbody>
</table>

By direct calculation\(^3\) 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, 

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

Now $x_3$ has additional importance: if $x_3 = t$, 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 $E_1 \land E_2$, we have

$$P[E_1, E_2] = P[E_1, E_2 \mid x_3 = t] P[x_3 = t] + P[E_1, E_2 \mid x_3 = f] P[x_3 = f].$$

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

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

\(^3\) $2 \cdot (49/64) + 7/64 + 7/64 = 7/4.$
Altogether we obtain

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

Likewise, one can compute \( P[E_1 \land \bar{E}_2] \), \( P[\bar{E}_1 \land \bar{E}_2] \), and \( P[\bar{E}_1 \land \bar{E}_2] \). 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) \( \frac{7}{4} \).

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, 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. Over \( m \) clauses \( C_1, \ldots, C_m \), there are just too many possible scenarios to calculate everything exactly. We have bumped into the following more general question: how can 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, then, 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 \( E[ \sum_{i=1}^m Y_i ] \).

As established above, it is easy to see that

\[ E[Y_i] = \frac{7}{8} \]

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

\[ E \left[ \sum_{i=1}^m Y_i \right] = \sum_{i=1}^m E[Y_i] = \frac{7}{8} m. \]
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 1.7.** For all $k \in \mathbb{N}$, there is a randomized $(1 - 1/2^k)$-approximation algorithm for $k$-SAT.

**Derandomization.** Having now obtained a randomized approximation algorithm for 3-SAT, we will use the randomized algorithm as an inspiration and guide for a deterministic algorithm with the same approximation ratio.

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, $\mathbb{E}[Z] = (7/8)m$.

Now, among many other decisions, a deterministic algorithm must decide whether to set $x_1 = t$ or $x_1 = f$. We now know that a uniformly random choice is pretty good on average. Loosely and 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

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

Rearranging, we have

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

That is, fixing either $x_1 = t$ or $x_1 = f$, 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 $\mathbb{E}[Z \mid x_1 = t]$ and $\mathbb{E}[Z \mid x_1 = f]$. Fortunately this is easy to do. Consider the case $x_1 = t$. By linearity of expectation we have

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

For a particular $Y_i$, we have
1. Randomized searching and sorting

1.4 Randomized sorting

Thus we can calculate $E[Z \mid x_1 = t]$ and similarly $E[Z \mid x_1 = f]$ 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 $a_1, \ldots, a_k \in \{t, f\}$ for $x_1, \ldots, x_k$, respectively, such that

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

Consider $x_{k+1}$. We have

$$E[Z \mid x_1 = a_1, \ldots, x_k = a_k] = \frac{1}{2} E[Z \mid x_1 = a_1, \ldots, x_k = a_k, x_{k+1} = t] + \frac{1}{2} E[Z \mid x_1 = a_1, \ldots, x_k = a_k, x_{k+1} = f],$$

hence setting either $x_{k+1} = t$ or $x_{k+1} = f$ 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 $a_1, \ldots, a_n$ such that

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

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.

1.4 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.

\footnote{For 3-SAT, $k_i = 2$ if $\bar{x}_i \in C_i$, and $k_i = 3$ otherwise.}
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 1.8.** 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 $i$th 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 $E[\sum_{i<j} X_{ij}]$.

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,

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

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

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

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. \qed
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.)

Our proof will require the following lemma.

**Lemma 1.9 (Markov’s inequality).** Let $X \geq 0$ be a nonnegative random variable, and $\alpha \geq 0$. Then

$$\Pr[X \geq \alpha] \leq \alpha / \mathbb{E}[X].$$

**Proof.** Since $X$ is nonnegative we have

$$\mathbb{E}[X] = \mathbb{E}[X \mid X \geq \alpha] \Pr[X \geq \alpha] + \mathbb{E}[X \mid X < \alpha] \Pr[X < \alpha] \geq \alpha \Pr[X \geq \alpha]$$

for all $\alpha \geq 0$. \qed

Note that Markov’s inequality is quite intuitive. Consider for example $\alpha = 2 \mathbb{E}[X]$. 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.

We now present the high probability bound for quick-sort.

**Theorem 1.10.** 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

$$\Pr[D_e \geq 32 \ln n] \leq 1/n^3.$$  \hfill (1.6)
Assuming (1.6) holds for any $e$, we then have

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

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 (1.6) 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 $\mathbf{E}[X_i] = 1/\text{poly}(n)$ 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

$$\mathbf{E}[X_1] = \frac{1}{2} \mathbf{E}[X_1 \mid \text{first pivot is good}] + \frac{1}{2} \mathbf{E}[X_1 \mid \text{first pivot is bad}]$$

$$\leq \frac{1}{2} \cdot \frac{3}{4} n + \frac{1}{2} n = \frac{7}{8} n.$$

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

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

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

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

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

26
Here, in (d), we applied the inequality \( 1 + x \leq e^x \) which holds for all \( x \). Finally, by Markov’s inequality, we have

\[
P[D_e \geq k] \leq P[X_k \geq 1] \leq E[X_k] \leq 1/n^3,
\]
as claimed in eq. (1.6). This completes the proof.

1.5 Additional notes and materials

Quicksort is also covered in [MotwaniRaghavan]. This particular proof of the high probability bound for quick-sort is from [SarielQuickSortNotes]. 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 2022 (undergraduate algorithms) 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.

1.6 Exercises

Additional exercises may be found it [MotwaniRaghavan].

**Exercise 1.1.** Prove or disprove: For any two events \( A, B \),

\[
P[A \land B] \leq \min\{P[A], P[B]\}.
\]
Exercise 1.2. Prove or disprove: For any two events $A$ and $B$, if $P[A] + P[B] > 1$, then $P[A \land B] > 0$.

Exercise 1.3. Let $A$ and $B$ be two events. Prove that the following three identities are all equivalent:


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

Exercise 1.4. Let $A$ and $B$ two events. Prove that $A$ and $B$ are independent iff $\bar{A}$ and $\bar{B}$ are independent.

Exercise 1.5. Prove linearity of expectation (theorem 1.6).

Exercise 1.6. Prove or disprove: for any two random variables $X$, $Y$, and real-valued function $f(X, Y)$, we have

$$E_X \left[ E_Y [f(X, Y) | X] \right] = E_Y \left[ E_X [f(X, Y) | Y] \right].$$


Exercise 1.8. 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 1.9. 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 1.10.** 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.

![Dice](image)

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.

![Dice](image)

**Exercise 1.11.** For $k \in \mathbb{N}$, 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? Prove your answer.

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 1.12.** Recall the quick-select algorithm introduced in section 1.1.3. The first goal of this exercise is to prove that quick-select takes $O(n)$ time in expectation. We ask you to prove this in two different ways which offer two different perspectives. Both analyses should use linearity of expectation and we ask you to point this out for both.

---

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 $p = 1/2$. 

---

29
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 $O(n)$. Use this to obtain a $O(n)$ expected running time, overall.

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 $k$th 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 $O(1)$.

(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 1.13.** 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

---

6Hint: Exercise 1.11.
$O(n^3)$ time and there are (more theoretical) algorithms with running time roughly $O(n^{2.37...})$. 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 $A(Bx)$ then $AB$.) 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 $1/2$.  
2. Use the preceding result to show that if $AB \neq C$, then with probability at least $1/2$, $ABx \neq Cx$.
3. Suppose we want to decrease our probability of error to (say) $2^{-n}$. Based on the algorithm above, design and analyze fast randomized algorithm with the following guarantees.
   - If $AB = C$, then it always reports that $AB = C$.
   - If $AB \neq C$, then with probability at least $1 - 2^{-n}$, it reports that $AB \neq C$.

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

Exercise 1.14. You have a sequence of $n$ switches $S_1, \ldots, S_n$ that jointly control $m$ light bulbs $L_1, \ldots, L_m$. 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”.

---

7 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 $S_{n-1} = \sum_{i=1}^{n-1} x_i y_{i-1}$ 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 $y_n = 0$?)

8 Even if you haven’t solved part 1 you may assume it to be true.

9 Note: I should have written “...probability of error to (say) $1/n^2$”, instead of $2^{-n}$, because the running time for $1/n^2$ error probability is more interesting. But I will leave it as it is since we are close to the deadline.
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.
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 \textit{sublinear} space.

**Streaming models.** We study the heavy hitter problem in the \textit{streaming model} of computation. In the streaming model, the input is a sequence of items presented to the algorithm \textit{one at a time}. We assume the algorithm has memory much less 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.

![Algorithm w/ limited space](image)

**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 \textit{absolute frequency} of an element \( e \), denoted \( f_e \), is the number of times the element appears in the stream. The \textit{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 total frequency divided by \( m \).

Ideally we could keep track of the exact frequencies of all the elements. This is impossible with less than \( \min\{n, m\} \) bits\(^2\). Here, and unlike standard algorithmic settings, \( O(m) \) or \( O(n) \) bits is not good enough. \textit{We have entered an algorithmic regime where simple and exact computations cannot be taken for granted.} Our first step, then, is to identify some new problems that are both \textit{tractable} and \textit{useful}. We may have to relinquish exactness and consider \textit{approximations}, where we allow for some error that we can analyze and control. And, last but not least, we will give up on deterministic computation, and design \textit{randomized} algorithms that have some small – analyzed and controlled – \textit{probability of failure}.

\(^2\text{One can formalize this impossibility as follows. There are } \binom{m+n-1}{n-1} \text{ ways to make a frequency} \)
\textbf{\(\epsilon\)-heavy hitters.} Given a fixed parameter \(\epsilon \in (0, 1)\), an \(\epsilon\)-heavy hitter is an element with relative frequency \(\geq \epsilon\). The heavy hitters problem is to identify all of the \(\epsilon\)-heavy hitters for an input parameter \(\epsilon > 0\). Note that there can only be \((1/\epsilon)\)-many \(\epsilon\)-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 \(\epsilon\). Equivalently, we want to estimate the absolute frequency of each element up to an \(\epsilon m\)-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 \(\epsilon\)-relative error, however, means that \(0\) is a satisfactory estimate for all but at most \(\lfloor 1/\epsilon \rfloor\) elements.

If we can estimate the absolute frequency of each element up to additive error \(\epsilon\), then one can find all \((3\epsilon)\)-heavy hitters by considering all of the elements with estimated value at least \(2\epsilon\). Such a list might also include \(1/\epsilon\) extra elements who have relative frequency \(< 3\epsilon\), but still have frequency \(> 1/\epsilon\). 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 \textbf{a priori} which of the elements are the \(\epsilon\)-heavy hitters, then \(\epsilon\)-frequency estimation is trivial. Namely, we would maintain a counter for each of the \([1/\epsilon]\)-many \(\epsilon\)-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 vector of \(n\) items where the total sum is \(m\). \(\text{(Why?)}\) 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 \( \binom{m+n-1}{n-1} \) 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

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

Thus

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

Slightly better low bounds can be obtained via \textit{Stirling’s approximation}, which is also related to entropy.
extra for safe measure). Although we do not know the heavy hitters, we will use randomized hash functions to obtain a similar effect.

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 will discuss hash tables in detail in the following chapter. Let us now define hash functions mathematically.

Loosely speaking, a hash function is a randomly constructed function $h : [n] \rightarrow [k]$ 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 3.1). 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 $h : [n] \rightarrow [k]$ is a uniformly random function $h : [n] \rightarrow [k]$. 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

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

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

$$P[h(i_1) = j_1, h(i_2) = j_2, \ldots, h(i_\ell) = j_\ell] = P[h(i_1) = j_1] P[h(i_2) = j_2] \cdots P[h(i_\ell) = j_\ell] = \frac{1}{k^\ell}.$$

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 \(h : [n] \rightarrow [k]\) is universal if for any distinct indices \(i_1 \neq i_2 \in [n]\), we have

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

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 \(h : [n] \rightarrow [k]\)

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

where \(p\) is a prime number larger than \(n\), \(a \in \{1, \ldots, p - 1\}\) is drawn uniformly at random, and \(b \in \{0, \ldots, p\}\) 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 \([n]\) coming in a stream of elements. We assume we know \(n\) a priori but not the length of the stream, which one can think of as being of infinite length. 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 \((\varepsilon m)\)-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 \(1/\varepsilon\) heavy hitters, this approach would
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 $h : \{1, \ldots, n\} \rightarrow \{1, \ldots, w\}$

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.

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.

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 $h : \{1, \ldots, n\} \rightarrow \{1, \ldots, w\}$. 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 $\epsilon m$ to $A[h(e)]$. Here the intuition is that the “noise” coming from other frequencies is spread out by the hash function over $\epsilon/2$ 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

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

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

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

38
2. Hashing and heavy hitters

2.4 Amplification

count-min(ε > 0, δ > 0)

1. build \( d = \lceil \log \frac{1}{\delta} \rceil \) instances \((A_1, h_1), \ldots, (A_d, h_d)\) of hashed-counters(ε) over the stream

2. to query an element \( e \):

   A. return \( \min_{i=1, \ldots, d} A_i[h_i(e)] \)

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 \( A[h(e)] − f_e ≥ 0. \)

Given that it is impossible to track frequencies exactly in sublinear space, it is surprising 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 \( \epsilon m \)-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 \( \frac{1}{2} = .5 \). With two coin tosses, the probability that both come up tails is still \( \frac{1}{4} = .25 \). But with 100 coin tosses, the probability that all 100 coin tosses come up tails is \( \frac{1}{2^{100}} \approx .000000000000000000000000000000007886... \).

The point is that independent trials magnify the probability of at least one 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

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

Proof. We have

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

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

For $\delta$ 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($\epsilon, 1/n^2$) 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($\epsilon, 1/n^2$) overestimates the total frequency of each element with additive error at most $\epsilon m$ 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($\epsilon, 1/n^2$) 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 $\Delta$, signifying that we should increase the frequency count for $e$, $f_e$ by $\Delta$. $\Delta$ 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 $\Delta$ to $A_i[h_i(e)]$ for each instance $(A_i, h_i)$ of hashed-counters. The additive error is now $\epsilon$ times the sum of all $\Delta$’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 expensive. 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-sketch** 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-sketch** does not give unbiased estimates of the counters. Instead, **count-min-sketch** 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 [Nelson2020; ChekuriCountMinNotes2014].

**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.
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 \( h : \mathbb{N} \rightarrow \mathbb{N} \) 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 \( h : \mathbb{N} \rightarrow \mathbb{N} \) as follows. First, let \( p \) be any prime number > \( n \). Draw \( a \in \{1, \ldots, p-1\} \) uniformly at random, and draw \( b \in \{0, \ldots, p-1\} \) uniformly at random. We define a function \( h(x) \) by

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

1. Let \( x_1, x_2 \in \mathbb{N} \) with \( x_1 \neq x_2 \), and let \( c_1, c_2 \in \{0, \ldots, p-1\} \) with \( c_1 \neq c_2 \). Show that the system of equations

\[
\begin{align*}
ax_1 + b &= c_1 \mod p \\
ax_2 + b &= c_2 \mod p
\end{align*}
\]

uniquely determines \( a \in \{1, \ldots, p-1\} \) and \( b \in \{0, \ldots, p-1\} \).

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

\[\text{Here it is helpful to know that division is well-defined on the set of integers modulo } p \text{ when } p \text{ is prime. More precisely, } \frac{a}{b} \text{ is defined as the unique integer } c \text{ such that } bc = a.\]

43
2. Let $x_1, x_2 \in [n]$ with $x_1 \neq x_2$, and let $c_1, c_2 \in \{0, \ldots, p - 1\}$ with $c_1 \neq c_2$. Show that

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

(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 $c_1 \in \{0, \ldots, p - 1\}$. Show that

$$\sum_{c_2 \in \{1, \ldots, p\}, \atop c_2 \neq c_1} P[ax_1 + b = c_1, ax_2 + b = c_2] \leq \frac{1}{pk}.$$ 

- The LHS represents $P[ax_1 + b = c_1 \mod p \text{ and } h(x_2) = h(x_1)]$.\footnote{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 $P[h(x_1) = h(x_2)] \leq \frac{1}{k}$.\footnote{Here we note that for $x_1 \neq x_2$, $ax_1 + b = ax_2 + b \mod p$ iff $a = 0$.}

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

1. Contains all of the $\epsilon$-heavy hitters.

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

Your space usage should be comparable to the space used by the count-min-sketch data structure.\footnote{Here the elements are integers from $[n] = \{1, \ldots, n\}$, where $n$ is known, and $\epsilon \in (0, 1)$ is an input parameter.}

\footnote{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.}

\footnote{You may want to use the $\text{count-min}(\epsilon, \delta)$ data structure as a black box, but you should be clear about your choice of parameters $\epsilon$ and $\delta$.}
**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 $\epsilon$-heavy hitters, and (b) only includes $(\epsilon/2)$-heavy hitters. The tricky part is that count-min-sketch only approximates the frequencies. You may want to account for the fact that an instance of count-min-sketch($\epsilon$, $\delta$) may overestimate the relative frequency of an element by as much as $\epsilon$, which can make a very infrequent element look like an $\epsilon$-heavy hitter.

**Exercise 2.4.** In this exercise, we develop a refined analysis that can reduce the additive error substantially in many real settings.

Let $S$ denote the sum of frequency counts of all elements that are not $\epsilon$-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 $w$ by a constant factor, and using $k$-wise independent hash functions for $k = O(1/\epsilon)$ (see page 55 for the definition of $k$-wise independent hash functions), one can estimate the frequency of every element with additive error at most $\epsilon S$ with high probability in $O(\log(n)/\epsilon)$ space.

**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 $k$th 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:

10 Hint: It might be helpful to think about the special case of $S = 0$.

11 Hint: $k$ was chosen to be greater than the maximum number of heavy hitters. Why?

12 Actually, there’s a way to do this with only universal hash functions. But you’re welcome to use $k$-wise independence if you find it convenient. You may first want to solve the problem assuming an ideal hash function, and figure out how to incorporate limited independence afterwards.
2. Hashing and heavy hitters

2.8. Exercises

Kent Quanrud
Fall 2022

---

**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. \( m \leftarrow 0, \ s \leftarrow \text{nil} \).

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 \( i \)th element in the stream. For \( m \in \mathbb{N} \) let \( s_m \) denote
the value of \( s \) after the \( m \)th iteration. Show that for all \( i \) and \( m \),

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

That is, for each \( m \), \( s_m \) is a uniformly random element out of \( \{e_1, \ldots, e_m\} \).

---

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
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.

---

13 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 \( m = i + 1 \)?

14 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,

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

In our argument, you may have two cases depending on whether or not \( e_m \in X \).
Chapter 3

Hash tables and linear probing

3.1 Dictionaries

Nowadays it is difficult to imagine programming without dictionaries and maps. These data structures are defined primarily by 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 \(i\)th 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.
3. Hash tables and linear probing

3.1. Dictionaries

Java Map Interface

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 len(dict) on a dictionary returns a list of all the keys used in the dictionary, in insertion order (if you want it sorted, just use set(dict) instead). To check whether a single key is in the dictionary use the in keyword. Here is a small example using a dictionary:

```python
>>> t = {'name': 'John', 'age': 30, 'city': 'New York'}
>>> t['name']
'John'
>>> t['city']
'New York'
```

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 size $n + 1$ and copying everything over. Even more problematic is the case where the keys are not neatly organized to be a 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 would not want to allocate an array so big. One could alternatively reindex the $n$ arbitrary keys into the slots 1,...,$n$. 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.
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 \texttt{get} and \texttt{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. The celebrated \textit{splay tree} data structure by \texttt{sleator-tarjan-85} readjusts itself with every \texttt{get} and \texttt{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 \textit{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 [\texttt{morrison-68}]), 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\footnote{Sometimes, log factors matter.}. They are more complicated to implement, and require a lot of \textit{pointer chasing}, which leads to many cache misses on the CPU.\footnote{This last point can be helped to some extent by cache-oblivious versions.}

We instead consider simpler \textit{randomized} approaches to the dictionary problem; namely, \textit{hash tables}. Hash tables combine the \textit{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 $h : [U] \rightarrow [m]$. 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 \textit{collide}, i.e., when $h(k') = h(k'')$ for two distinct keys $k'$ and $k''$. There are various ways, sometimes simple and sometimes clever, to account for collisions, such as the following.

1. Make $\ell$ so large that even a single collision is unlikely. Exercise 3.1 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 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 3.3.

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. We 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 negligible. 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 [karger+97]. Distributed dictionaries are particularly useful for caching on the web.
3. Hash tables and linear probing

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 $h : [U] \rightarrow [m]$. Here we analyze the setting where $h$ is a universal hash function, but later we will also explore stronger notions of independence. Exercise 3.2 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 $h : [n] \rightarrow [m]$ is universal if, for any two indices $i_1 \neq i_2$, we have

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

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 \( A[1,\ldots,m] \), 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

\[
E[|k' : h(k') = h(k)|] \equiv \sum_{k' \neq k} P[h(k') = h(k)] \equiv \sum_{k' \neq k} \frac{1}{m} \equiv \frac{n-1}{m}
\]

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

### 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 1964 [knuth]. Knuth has been called the “father of the analysis of algorithms”, and he is credited with formalizing the subject and popularizing $O$-notation. As Knuth tells it, this was the first algorithm he ever formally analyzed, and therefore, arguably, the first algorithm that anyone has ever (so) formally analyzed. He showed that for ideal hash functions,

---

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.
the expected time of any operation is $O((n/(m - n))^2)$; in particular, a constant, whenever $m$ is bigger than $n$ 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.

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

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

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. We note that the hash function

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \mod p,$$

where $p$ is a prime number larger than $U$, and $a_0, \ldots, a_{k-1} \in [p]$ are sampled independently and uniformly at random, is a $k$-wise independent hash function.

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, ss-89; ss-90 showed that $O(\log n)$-wise independence sufficed\(^6\). Then, in 2007, ppr-09 showed that (just!) 5-wise independence sufficed. This was dramatic progress for arguably the oldest problem in algorithm design. Soon after, [patrascu-thorup-16] showed that 4-wise independence was not enough. So the answer is 5!

Here we give a simplified analysis of the result of [ppr-09] based on ideas in [patrascu-thorup-16]. 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 [ppr-09] and also the reader is encouraged to refine the analysis themselves. Similar proofs of the constant time bound can be found in [thorup-survey; nelson-linear-probing-video].

### 3.3.1 Technical preliminaries: 4-wise independence

Before proceeding, we state a probabilistic inequality that we require. Here we will limit ourselves to a definition and the lemma statement; a more detailed discussion and the proof is given in section 3.4.

---

\(^6\)Alan Siegel taught me algorithms.
Definition 3.4. A collection of $n$ variables $X_1, \ldots, X_n$ is $k$-wise independent if for any $k$ variables $X_{i_1}, \ldots, X_{i_k}$, and values $y_1, y_2, \ldots, y_k$, we have

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

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

$$P\left[\sum_{i=1}^{n} X_i \geq \mu + \beta\right] \leq \frac{4\mu^2}{\beta^4}. \tag{3.2}$$

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

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

Lemma 3.5 says that

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

Compare the RHS of (3.1) with the RHS of (3.2). In (3.2), the upper bound is decreasing in $\alpha$ at a $1/\alpha^4$ rate, compared $1/\alpha$ in (3.1). Moreover, (3.2) is decreasing in the expected value $\mu$ 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.

3.3.2 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. 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”.

55
3. Hash tables and linear probing

3.3. Linear probing

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, and \( x \) occupies a cell in the run containing \( A[h(x)] \). Therefore each operation with an item \( x \) takes time at most proportional to the length of the run containing \( x \).

Let \( i = h(x) \), and let \( R \) be the run at index \( i \). Note that \( R \) and its length \( |R| \) are random. We have

\[
E\left[ \text{running time (up to constants)} \right] \leq E[|R|] = \sum_{\ell=1}^{n} \ell P[|R| = \ell] \leq \sum_{k=1}^{\lceil \log n \rceil} 2^k P\left[ 2^{k-1} < |R| \leq 2^k \right]. \quad (3.3)
\]

For each \( k \in \mathbb{N} \), let

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

be the interval of length \( 2^{k+1} - 1 \) centered at \( i \).

If \( R \) has length \( |R| < 2^k \), and contains \( i \), then \( R \) must be contained in \( I_k \). Moreover, if \( R \) has length \( |R| > 2^{k-1} \), then at least \( 2^{k-1} \) items other than \( x \) hash to \( R \). Thus for each \( k \), we have

\[
P\left[ 2^{k-1} < |R| \leq 2^k \right] \leq P\left[ \text{at least } 2^{k-1} \text{ other} \right].
\]

Since \( h \) is 5-wise independent, conditional on \( h(x) = i \), the remaining hash values are 4-wise independent, and each lands in \( I_k \) with probability \( p = |I_k|/m \). Let

\[
\mu = E\left[ \text{# other items hashing into } I_k \right].
\]

We have

\[
\mu = \frac{|I_k|n (\omega)}{m} \leq 2^{k-2}.
\]
where (a) is because \( m \geq 8n \). We have

\[
P\left(\text{# other items hashing into } I_k \right) > 2^{k-1} \leq \frac{4 \max\{\mu, \mu^2\}}{(2^{k-1} - \mu)^4} \leq \frac{4(2^{k-2})^4}{(2^{k-2})^4} \leq \frac{1}{2^{2k-6}}.
\]

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

\[
E\left[\text{running time}\right] \leq \text{RHS}(3.3) \leq \sum_{k=1}^{[\log n]} 2^k \cdot \frac{1}{2^{2k-6}} = 2^6 \sum_{k=1}^{[\log n]} \frac{1}{2^k} \leq 2^6.
\]

A constant! \( \square \)

### 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 \( X_1, \ldots, X_k \) be \( k \)-wise independent random variables. Then

\[
E[X_1X_2 \cdots X_k] = E[X_1]E[X_2] \cdots E[X_k].
\]

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, \( X_1, \ldots, X_k \) are mutually independent. The probability that \( k \) independent coin tosses all comes up heads is \( 1/2^k \), so \( E[X_1 \cdots X_k] = 1/2^k \).
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 $X_1, \ldots, X_k = 1$ is the probability of a single coin coming up heads, $1/2$, and so $E[X_1 \cdots X_k] = 1/2$.

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

**Proof of lemma 3.7.** We have

\[
E[X_1X_2 \cdots X_k] \\
\overset{(a)}{=} \sum_{y_1, y_2, \ldots, y_k} y_1 y_2 \cdots y_k P[X_1 = y_1, X_2 = y_2, \ldots, X_k = y_k] \\
\overset{(b)}{=} \sum_{y_1, y_2, \ldots, y_k} y_1 y_2 \cdots y_k P[X_1 = y_1] P[X_2 = y_2] \cdots P[X_k = y_k] \\
= \left( \sum_{y_1} y_1 P[X_1 = y_1] \right) \left( \sum_{y_2} y_2 P[X_2 = y_2] \right) \cdots \left( \sum_{y_k} y_k P[X_k = y_k] \right) \\
\overset{(c)}{=} E[X_1] E[X_2] \cdots E[X_k].
\]

Here (a) is by definition of expectation\(^7\). (b) is by $k$-wise independence. (c) is by definition of expectation, for each $X_i$. \(\square\)

### 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 $i$, $E[X_i] = p$. Let $\mu = pn = E[\sum_{i=1}^{n} X_i]$. Then for any $\beta > 0$,

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

**Proof.** We have

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

\(^7\)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, $P[X_1 = y_1, \ldots, X_k = y_k]$. 

58
The key step is (a), where we raise both sides to the fourth power. (b) is by Markov’s inequality. We claim that

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

which would complete the proof. We first have

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

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

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

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

$$
E [(X_i - p)^4] = p(1-p)^3 + (1-p)p^3 \leq p(1-p).
$$

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

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

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

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

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,

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

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

59
Plugging back in above, we have
\[
\mathbb{E} \left[ \left( \sum_{i=1}^{n} X_i - \mu \right)^4 \right] = np(1 - p) + \binom{n}{2} \binom{4}{2} \left( p^2(1 - p)^2 \right) \leq np + 3(np)^2,
\]
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

Thorup2015 describes several families of hash functions with both theoretical and practical considerations. See [EricksonHashingNotes] 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 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

\[
P[\text{at least } \ell \text{ items being hashed to } A[i]] \leq \frac{1}{\ell!}.
\]

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, with \( k \geq n \). Show that for \( k \geq n^2 \), \( h \) has no collisions with probability \( \geq 1/2 \).

\(^8\)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.
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 array 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 $i$th 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

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

4. Show that

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

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

This chapter is about applying random sampling while trying to preserve combinatorial structures, like graphs.

Consider, for example, the \((s,t)\)-flow problem. We have as input a graph \(G = (V,E)\), positive edge capacities \(c : E \to \mathbb{R}_{>0}\), and two vertices \(s\) and \(t\). We want to route the maximum amount of flow from \(s\) to \(t\). Can we sample a small subgraph of \(G\) while preserving the value of the maximum \(s\) to \(t\) flow? If so, then we could run a max flow algorithm on the sampled subgraph and get faster overall running times. Try to imagine sampling edges from a graph in the interest of flow. Flow has a combinatorial aspect that would appear much more delicate than, say, heavy hitter estimates. For example, missing a single important edge when sampling can disrupt a polynomial number of paths used by the maximum flow. Nonetheless, we will see that for undirected graphs, the surprising answer is yes: one can indeed subsample the important parts of an undirected graph and preserve the maximum flow.

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 minimize

\[
\sum_{e \in C} c(e) \quad \text{over all cuts } C \subseteq E.
\]  

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 (4.1).
4. Sampling edges

4.1. Minimum cut

We will study a subtle algorithm discovered by karger-93, 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 4.1).

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 $T - e$.

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
4. Sampling edges

4.1. Minimum cut

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-93.

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 $c/e : E/e \rightarrow \mathbb{R}_{>0}$ denote the corresponding capacities. Figure 4.3 sketches a few
iterations of the algorithm applied to a barbell graph.

The intuition behind \texttt{random-contractions} is as follows. Here we describe the
intuition for 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

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

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

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 iteration, assuming $C$ survived the first iteration, $C$ has (at most) a $2/(n - 1)$ chance of being hit. Continuing in this fashion, assuming $C$ survived the first $i - 1$ iterations, $C$ has a $2/(n - i + 1)$ chance of being hit in the $i$th 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 above argument, as well as extend it to positive capacities. For ease of notation, for a set of edges $C \subseteq E$, we denote the sum of
4. Sampling edges
4.1. Minimum cut

capacities over \( C \) by

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

**Lemma 4.1.** Let \( C^* \) be the minimum cut in \((G, c)\), and suppose \( e \notin C \). Then \( C^* \) 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 \( \Pr[e \in C^*] \leq \frac{2}{n} \).

**Proof.** We have \( \Pr[e \in C^*] = \frac{\sum_{e \in C^*} c(e)}{\sum_{e \in E} c(e)} \leq \frac{2}{n} \) by (a) lemma 4.2.

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

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

\[
\Pr[E_k \mid E_{k-1}] \geq 1 - \frac{2}{n-(k-1)} \text{ for each } k \in [n].
\]

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

\[
\Pr[E_{n-2}] = \prod_{k=1}^{n-2} \Pr[E_k \mid E_{k-1}] \geq \prod_{i=3}^{n} \left( 1 - \frac{2}{i} \right) = \prod_{i=3}^{n} \frac{i-2}{i} = \frac{(n-2)!2}{n!} = \frac{1}{\binom{n}{2}}.
\]

\( \square \)
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.

**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-stein** 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-stein** proposed randomly contract edges for a fixed number of iterations so that 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
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-stein.

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 $C^*$, 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 $C^*$ after $i$ iterations. We ware interested in $P[E_{n-k}]$. We have

\[
P[E_{n-k}] = \prod_{i=1}^{n-k} P[E_i \mid E_{i-1}] = \prod_{i=k+1}^{n} \left(1 - \frac{2}{i}\right) = \frac{\prod_{i=3}^{k} \frac{i-2}{i}}{\prod_{i=3}^{k} \frac{i-2}{i}} = \frac{k(k-1)}{n(n-1)}.
\]

by (a) lemma 4.3.\hfill\square

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

69
4. Sampling edges
4.2. Amplification by branching

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 \( \varphi \) \( i \) 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

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

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

![Randomized branching diagram]

**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

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

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

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

The first three values are

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

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

$$f(x) = x\left(1 - \frac{x}{4}\right)$$

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

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

The last inequality (a) is obtained by

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

\[\square\]

4.4 Sparsification

*Sparsification* refers to the general method of taking a large, dense graph and producing a smaller, sparse graph (over the same vertex set) that preserves some desired structure of the original graph.

We start with a very familiar example. Recall that two vertices $s$ and $t$ are connected in an undirected graph if there is a path from $s$ to $t$. Given an undirected graph $G = (V, E)$, suppose we wanted a sparse subgraph $G' = (V, E')$ such that any two vertices $s, t \in V$ are connected in $G$ iff they are connected in $G'$. Here there is a solution $G'$ with (slightly less than) $n$ edges. The reader probably knows the answer and should pause to realize it.

4.4.1 Preserving small connectivities

**Definition 4.10.** Let $G = (V, E)$ be an undirected graph. The connectivity between two vertices $s, t \in V$ is the size of the minimum $\{s, t\}$-cut. When $G$ is unweighted, this is also the maximum number of disjoint paths from $s$ to $t$. The connectivity of an edge $e \in E$ is defined as the connectivity of its endpoints.

Above, we discussed sparsifiers that maintain all pairwise connectivities in an undirected graph up to connectivity 1. Here we will analyze a generalization by *nagamochi-ibaraki-92-sparse* for maintaining connectivities up to a fixed cardinality $k \in \mathbb{N}$. That is, given an undirected graph $G = (V, E)$ and a parameter $k$, we will compute a set $F \subseteq E$ with less than $kn$ edges with the following guarantees.

1. If $s, t \in V$ have connectivity $\ell \leq k$ in $G$, then $s$ and $t$ have connectivity $\ell$ in $F$. 

72
2. If \( s, t \in V \) have connectivity \( \ell \geq k \) in \( G \), then \( s \) and \( t \) have connectivity \( \geq k \) in \( G \).

**nagamochi-ibaraki-92-sparse** proposed the following simple greedy algorithm. Consider the partition of \( E \) into forests \( F_1, F_2, F_3, \ldots \) constructed as follows.

> For each edge \( e \in E \) (in any order), find the first forest \( F_i \) such that \( F_i + e \) is also a forest, and add \( e \) to \( F_i \).

For each \( k \in \mathbb{N} \), let \( F_k = F_1 \cup \cdots \cup F_k \) denote the union of the first \( k \) forests, as a subgraph of \( G \).

**Theorem 4.11 (nagamochi-ibaraki-92-sparse).** Let \( s, t \in V \) have connectivity \( \ell \) in \( G \). Then for all \( k \leq \ell \), \( s \) and \( t \) have connectivity \( \geq k \) in \( F_k \).

The key observation to proving theorem 4.11 is the following.

**Lemma 4.12.** Let \( C = \delta(S) \) be an induced cut with \( \ell \) edges. Then for all \( k \leq \ell \), \( |C \cap F_i| \geq i \).

**Proof.** If not, there is an edge \( e \in C \setminus F_k \). For each \( i \in [k] \), if \( e \notin F_i \), then there must be another edge in \( C = \delta(S) \) in \( F_i \). Applied to each \( i \), we see that \( C \) has at least one edge in each \( F_i \), hence at least \( k \) total in \( F_k \) - a contradiction. \( \square \)

The proof of theorem 4.11 is now immediate: by lemma 4.12, every \( \{s, t\} \)-cut in \( F_k \) has at least \( k \) edges.

Theorem 4.11 gives us our first nontrivial sparsifier. With \( < kn \) edges, we can preserve all connectivities up to size \( k \). Don’t be fooled by its simplicity! This is a landmark algorithm that has inspired many more ideas.

In the sequel, we will be interested in sparsifiers whose size is independent of \( k \). To this end, we will have to introduce both approximations and randomization. Before continuing on, however, we require the following observation about the Nagamochi-Ibaraki sparsifier: Nagamochi-Ibaraki sparsifier encodes, for every edge \( e \), a lower bound on its connectivity, as follows.

**Lemma 4.13.** For each edge \( e \), if \( e \in F_k \), then \( e \) has connectivity \( \geq k \).

**Proof.** Let \( e = \{s, t\} \), let \( C \) be an \( \{s, t\} \)-cut, and let \( \ell = |C| \). If \( \ell < k \), then by lemma 4.12, all \( \ell \) edges of \( C \) would be in \( F_\ell \), a contraction to \( e \in F_k \). \( \square \)
4.4.2 Randomized sparsification

Let $G = (V, E)$ be an unweighted and undirected graph, and let $F_1, F_2, \ldots$ partition $E$ into forests per the greedy process in section 4.4.1. Let $\epsilon > 0$ be fixed.

Consider the randomized and reweighted subgraph $G' = (V, E', c')$ constructed as follows. For each edge $e \in E$, let $i$ be the index of the forest $F_i$ such that $e \in F_i$, and let

$$p_e = \min \left\{ \left( \frac{c \log^2(n)}{\epsilon^2} \right)^{\frac{1}{i}}, 1 \right\}$$

for a universal constant $c \geq 1$ TBD. The random graph $G' = (V, E', w)$ is now generated as follows.

Independently for each edge $e \in E$, with probability $p_e$, add $e$ to $E'$ with weight $w(e) = 1/p_e$.

We say that $(G' = (V, E'), w)$ preserves the weight of a cut $C = \delta(S)$ up to a $(1 \pm \epsilon)$-multiplicative factor if

$$(1 - \epsilon)|C| \leq \sum_{e \in C \cap E'} w(e) \leq (1 + \epsilon)|C|.$$ 

**Theorem 4.14.** With high probability, $(G', w)$ has $O\left( n \log^3(n)/\epsilon^2 \right)$ edges, and preserves the weight of every (induced) cut up to an $(1 \pm \epsilon)$-multiplicative factor.

We can interpret theorem 4.14 as a randomized approximation of the deterministic Nagamochi-Ibaraki sparsifiers from section 4.4.1. Recall that the Nagamochi-Ibaraki forests preserved all cuts up to a fixed cardinality $k$ exactly, with at most $k(n - 1)$ edges total. This bound is ideal for constant $k$, but less appealing for larger values of $k$. theorem 4.14 preserves all cuts of all sizes up to an $(1 \pm \epsilon$-multiplicative. The output graph is almost linear in the number of vertices. A natural application of theorem 4.14 is as a preprocessing step to maximum flow. By theorem 4.14, we can reduce the number of edges in the graph to $O\left( n \log^2(n)/\epsilon^2 \right)$ edges while preserving the value of the maximum flow up to an $(1 \pm \epsilon)$-multiplicative factor. Running Ford-Fulkerson (or whatever other flow algorithm) in $G'$ is faster because $G'$ is smaller, and will still give an accurate estimate of the flow.

We note that there are also algorithms for weighted graphs that are almost based on random sampling in proportion to some measure of connectivity; see section 4.4.3 below.
First observations

It is not at all obvious that $G'$ should approximately preserve the value of every cut. However, it is easier to see that $G$ is sparse.

Lemma 4.15. With high probability, $G'$ has $O\left(n \log^3(n)/\epsilon^2\right)$ edges.

Proof. Each edge in $F_i$ is added with probability $\leq O\left(\log^2(n)/\epsilon^2\right)$. In particular, there are at most $n-1$ edges with $p_e = c \log^2(n)/\epsilon^2$, $n-1$ edges with $p_e = c \log^2(n)/2\epsilon^2$, and so forth. We conclude that

$$
E[|E'|] = \sum_{e \in E} p_e \leq \frac{c \log^2(n)}{\epsilon^2} \sum_i \frac{1}{i} |F_i| \leq O\left(\frac{n \log^2(n)}{\epsilon^2}\right).
$$

For (a), we note that an easier upper bound is obtained by

$$
\sum_i \frac{1}{i} |F_i| \leq n \sum_i \frac{1}{i} = O(n \log(n)).
$$

To instead upper bound by $O(n)$, we observe that subject to $\sum_i |F_i| \leq m$, the LHS is maximized by have the first $O(m/n)$ have $n-1$ edges. \hfill \Box

Let us now return to cuts. We first verify that the random graph preserves the value of every cut in expectation.

Lemma 4.16. Let $C = \delta(S)$ be a cut. Then

$$
E \left[ \sum_{e \in C \cap E'} w(e) \right] = |C|.
$$

Proof. The key is that whenever we sample an edge $p_e$ with probability $p_e$, we scale up the capacity by $1/p_e$. These two factors cancel each other out. Indeed,

$$
E \left[ \sum_{e \in C \cap E'} w(e) \right] \overset{(a)}{=} \sum_{e \in C} P[e \in E'] E[w(e) \mid e \in E'] = \sum_{e \in C} p_e \cdot \frac{1}{p_e} = |C|.
$$

Here (a) is by linearity of expectation. \hfill \Box

We can also show that any particular cut is preserved with high probability.
Lemma 4.17. Let $C = \delta(S)$ be a cut. Then for any $\beta > 0$,

$$
P\left[\sum_{e \in S \cap E'} w(e) \geq (1 + \epsilon)|S| + \beta\right] \leq e^{-c\log^2(n)\beta/|S|}
$$

and

$$
P\left[\sum_{e \in S \cap E'} w(e) \leq (1 - \epsilon)|S| - \beta\right] \leq e^{-c\log(n)\beta/|S|}.
$$

Proof. Let $k = |C|$. Then $C \subseteq F_1 \cup \cdots \cup F_k$. For every edge $e \in C$, we either have $p_e = 1$ and $e \in C$ deterministically, or with probability $p_e$, $e \in E'$ and

$$
w(e) = \frac{1}{p_e} \leq \frac{\epsilon^2 ck}{\log^2(n)} = \left(\frac{\epsilon^2}{c\log^2(n)}\right)|C|.
$$

Taking a step back, our goal is to estimate $|C|$ by random sampling. As observed in lemma 4.16, the expected value of the random sampling is correct. Each edge $e \in C$ represents an independent random quantity that contributes at most a $(\epsilon^2/c\log^2(n))$-fraction of $C$. When many small and independent parts add up to a (relatively) large expected sum, then the random sum will be strongly concentrated around its concentration. Applying Chernoff bounds gives the desired result.

Note that for $\beta = \epsilon|S|$, lemma 4.17 says that any particular cut is preserved up to an $(1 \pm 2\epsilon)$-multiplicative factor with probability of error $\leq n^{-c\log(n)}$. This is very encouraging, as the error probability is small enough to take the union bound over polynomially many cuts. However, it is not enough for theorem 4.14, which says that all cuts are preserved with high probability – and there are, alas, $2^n$ many different cuts.

Overview of the proof

We have already collected some observations that suggests theorem 4.14 is not unreasonable. Every cut is preserved in expectation, and up to an $(1 \pm \epsilon)$-approximation with very high probability. The problem is that we want to preserve all cuts, there are too many cuts to simply apply a union bound.

We take inspiration from section 4.1, where we analyzed a simple randomized algorithm for minimum cut. Intuitively, when sparsifying a graph, the minimum cuts should be the hardest cuts to preserve as they have the smallest margin for error. A remarkable consequence of the contraction algorithm is that there are only $\binom{n}{2}$
minimum cuts in a graph. An extension of the same argument shows that, for any approximation factor $\alpha > 1$, there are at most $n^{O(\alpha)}$ $\alpha$-approximate minimum cuts. Thus there are only $\text{poly}(n)$ cuts within a factor of, say, 2 of the minimum cut. This structure paints a more optimistic picture then assuming that there are $2^n$ cuts we have to preserve and that they are all equally sensitive to random sampling.

We first bucket the edges by their sampling probabilities. For $k = 1, 2, \ldots$, let

$$H_k = \bigcup_{2^{k-1} \leq i \leq 2^k} F_i.$$ 

Note that all edges in a bucket $H_k$ are sampled with roughly the same probability, varying by at most a multiplicative factor of 2. This generates at most $2 \log(n)$ buckets total.

Fix a bucket $H_k$. In the foregoing, a section of a bucket $H_k$ is an edge set of the form $A = H_k \cap C$, where $C = \delta(S)$ is an induced cut. We want to show that each section $A = H_k \cap C$ is approximated fairly well by the sampled edges in $H_k$. To this end, one first observes that if $C \cap H_k \neq \emptyset$, then $C$ must have at least $2^{k-1}$ edges. Each edge $e \in H_k$ is given weight (roughly) $\epsilon^2 2^k / \log^2(n)$, which in particular is at most a $\epsilon^2 / \log^2(n)$ fraction of $|C|$ (up to constants). That is, each edge $e \in H_k \cap C$ represents a small and independently random contribution to the final estimate of $|C|$. One can apply concentration bounds and obtain the following.

**Lemma 4.18.** Let $k \in \mathbb{N}$. For every section of the form $A = H_k \cap C$, where $C = \delta(S)$ is a cut, we have

$$\mathbb{P}\left[ \sum_{e \in A \cap E'} w(e) \geq (1 + \epsilon)|A| + \left( \frac{\epsilon}{\log n} \right) |C| \right] \leq n^{-c_0 |C|/2^{k+1}}$$

and

$$\mathbb{P}\left[ \sum_{e \in A \cap E'} w(e) \leq (1 - \epsilon)|A| - \left( \frac{\epsilon}{\log n} \right) |C| \right] \leq n^{-c_0 |C|/2^{k+1}}.$$

We have lemma 4.18 as an exercise, although really all of the ideas of the proof are given in the description above.

Lemma 4.18 is helpful, but we find ourselves in a similar predicament as before, where now the number of sections of cuts may still be exponentially large. We want to argue that there is a nicer structure for which lemma 4.18 will suffice.

Intuitively speaking, we should worry most about the sections $A = H_k \cap C$ where $|C|$ is small, because these give us the smallest margin of error. We will argue,
by simulating the random contraction algorithm with some modifications based on Mader’s splitting lemma (introduced later), that the number of sections of the form \( H_k \cap C \) where \(|C| \leq 2^k\) is a polynomial. In fact, there is a fairly smooth increase in the number of distinct sections \( H_k \cap C \), where \(|C| \leq \alpha 2^k\) for \( \alpha > 1 \), as follows.

**Lemma 4.19.** Let \( H \) be a set of edges where every edge has connectivity \( \geq K \). Consider all sections of the form \( A = C \cap H \), where \( C = \delta(S) \) is a cut. For all \( \alpha \geq 1 \), there are at most \( n^{2\alpha} \) nonempty sections of the form \( A = H \cap C \), where \( C \) is a cut with \( \leq \alpha K \).

We will prove lemma 4.19 in section 4.4.2 below.

Lemma 4.19 implies a structure to the sections of a bucket \( H_k \) that is much more forgiving than simply assuming where \( 2n \) sections with cuts and every cut has size roughly \( 2^k \). Lemma 4.19 says that there are only polynomially many sections with cuts with \( \leq 2^k \) that we really need to be worried about. It also says that there are only polynomially many more sections of cuts with \( \leq 2^k+1 \) edges. While there are some more sections in this category, we also have twice as much room for error.

**Lemma 4.20.** Let \( k \in \mathbb{N} \). With probability \( \geq 1 - 1/\text{poly}(n) \), for all sections of the form \( A = H_k \cap C \), we have

\[
(1 - \epsilon)|A| - \frac{\epsilon}{\log n} |C| \leq \sum_{e \in A \cap E'} w(e) \leq (1 + \epsilon)|A| + \frac{\epsilon}{\log n} |C|.
\]

Let us prove lemma 4.20 now under the assumption that lemma 4.18 and lemma 4.19 hold true.

**Proof of lemma 4.20.** Recall that every edge \( e \in H_k = F_{2^k-1} \cup F_{2^k-1+1} \cup \cdots \cup F_{2^k-1} \) has connectivity \( \geq 2^{k-1} \). Thus any cut \( C = \delta(S) \) intersecting \( H_k \) has at least \( 2^{k-1} \) edges.

We first group the number of distinct sections \( H_k \cap C \) by \(|C|\). For \( \ell \in \mathbb{N} \), let

\[
\mathcal{A}_\ell = \{ H_k \cap C : 2^{k-\ell} \leq |C| < 2^{k+\ell} \}.
\]

Fix \( \ell \in \mathbb{N} \). By lemma 4.19, we have

\[
|\mathcal{A}_\ell| \leq n^{c_02^\ell} \tag{4.4}
\]

for some constant \( c_0 > 0 \). For each section \( A = H_k \cap C \) where \( C \in \mathcal{A}_\ell \), we have

\[
(1 - \epsilon)|A| - \frac{\epsilon}{\log n} |C| \leq \sum_{e \in A \cap E'} w(e) \leq (1 + \epsilon)|A| + \frac{\epsilon}{\log n} |C| \tag{4.5}
\]
4. Sampling edges

4.4. Sparsification

with probability of error
\[ \leq e^{-c \log(n) |C|} = n^{-c_1 |C|/2^k} \leq n^{-c_1 2^k/2} \]
for some constant \( c_1 > 0 \) that can be increased by increasing the constant \( c \) in our sampling probabilities \( p_e \). In particular, we can make (say) \( c_1 \geq 2c_0 + 4 \). In that case that probability of error is
\[ \leq n^{-(c_0+2)2^k}. \]
(4.6)

Comparing (4.4) and (4.6), we see that we can take the union bound over \( A_\ell \!). Indeed, by the union bound, the error probability of failing (4.5) for at least one \( A \in A_\ell \) is
\[ |A_\ell| \cdot n^{-(c_0+2)2^\ell} \leq n^{2^{\ell+1}}. \]

We can now take the union bound over all \( O(\log \log n) \) choices of \( \ell \).

Theorem 4.14 now follows from lemma 4.20. We first restate theorem 4.14 for the reader’s convenience.

**Theorem 4.14.** With high probability, \((G', w)\) has \( O\left(n \log^2(n)/\epsilon^2\right) \) edges, and preserves the weight of every (induced) cut up to an \((1 \pm \epsilon)\)-multiplicative factor.

**Proof.** We have already verified the number of edges in lemma 4.15. It remains to verify each cut. With high probability, lemma 4.20 holds for every bucket \( H_k \). For each cut \( C = \delta(S) \), we have
\[ \sum_{e \in C \cap E'} w(e) = \sum_k \sum_{e \in C \cap H_k \cap E'} w(e) \leq \sum_k \left( (1 + \epsilon)|C \cap H_k| + \frac{\epsilon}{\log n} |S| \right) = (1 + 2\epsilon)|C|. \]
Likewise, we have
\[ \sum_{e \in C \cap E'} w(e) = \sum_k \sum_{e \in C \cap H_k \cap E'} w(e) \geq \sum_k \left( (1 - \epsilon)|C \cap H_k| - \frac{\epsilon}{\log n} |S| \right) = (1 - 2\epsilon)|C|. \]
This completes the proof of theorem 4.14.

**Counting sections**

It remains to prove lemma 4.19, regarding the number of distinct sections induced by cuts of various sizes. To prove lemma 4.19, we require the following beautiful graph theoretic fact due to Mader and Lovasz. We will not prove this fact, and instead refer the reader to the textbook [Frank].
Fact 4.21 (Splitting off lemma). Let $G = (V, E)$ be an undirected graph, and let $v$ be a fixed vertex with even degree $k$. Then one “split off” the edges at $v$ while preserving the connectivity of all pairs of vertices (not including $v$) in the graph.

Here “splitting off” means that we can pair up the edges incident to $v$, and replace them with the $k/2$ edges formed by concatenating them and removing $v$ from the middle. (e.g., two edges $\{a,v\}$ and $\{b,v\}$ are “split off” and replaced by a new edge $\{a,b\}$.)

To get some sense of our use of fact 4.21, fix $k \in \mathbb{N}$, and consider a section of the form $A = C \cap H_k$ where $C = \delta(S)$. Suppose we apply the splitting off lemma to split off a vertex $v$ with degree $< 2^{k-1}$. First, note such a vertex $v$ cannot be incident to any edge in $H_k$ because every edge in $H_k$ still has connectivity $\geq 2^{k-1}$. Second, when splitting off $v$, we note that the cut $C$ might decrease in size, but cannot increase. Third, the connectivity of every edge in $H_k$ is preserved, and in particular stays $\geq 2^{k-1}$. Thus when splitting off $v$, the section $A = C \cap H_k$ maps directly to the same section $A$, now as an intersection $w \cap H_k$ where $|w| \leq |C|$.

We now restate and prove lemma 4.19.

Lemma 4.19. Let $H$ be a set of edges where every edge has connectivity $\geq K$. Consider all sections of the form $A = C \cap H$, where $C = \delta(S)$ is a cut. For all $\alpha \geq 1$, there are at most distinct $n^{2\alpha}$ nonempty sections of the form $A = H \cap C$, where $C$ is a cut with $|C| \leq \alpha K$.

Proof. We will prove the bound indirectly by directly proving the following: the distinct number of sections of the desired form in an $n$-vertex graph is at most the maximum number of $\alpha$-approximate minimum cuts in an $n$-vertex graph.

Suppose we run the contraction algorithm with the following two modifications.

1. We first note that by doubling all the edges initially, we can assume the graph is even degree. The evenness of the degree is also preserved under edge contractions. Now, whenever there is a vertex $v$ with degree $< K$, we split off $v$.

2. We stop the algorithm when there are $\lceil 2\alpha \rceil$ vertices left in the graph, and return all sections induced by subsets of these vertices. (Thus $< 2^{\lceil 2\alpha \rceil}$ sections are returned.)

Fix any section of the form $A = H \cap C$, where $|C| < \alpha K$. Note that $A$ is preserved under splitting off, as well as the fact that $A$ can be represented as an intersection $H \cap C$ with a cut $C$ such that $|C| < \alpha 2^K$. We return $A$ iff (some) cut $C$ such that $A = H \cap C$ survives to the end of the algorithm.
Recall that the splitting off ensures that the minimum degree is $\geq K$ at all times. If there are $n$ vertices left and the minimum degree is $K$, then the probability that the next edge is sampled from $C$ is

$$\leq \frac{2|C|}{Kn} < \frac{2\alpha}{n}.$$ 

This is the exact same as if counting the number of $\alpha$-approximate minimum cuts in $\Phi$. Thus we obtain the same bound.

4.4.3 Weighted, Sparser, Stronger... and Deterministic

benczur-karger showed how to compute cut sparsifiers for weighted graphs with only $O(n \log(n)/\epsilon^2)$ edges. This approach samples edges in proportion to (underestimates of) their edge strengths, a different notion of connectivity. [benczur-karger] was actually the first such sparsification result, and precedes the simpler algorithm described above, which is from flhp. We point out two generalizations of cut sparsifiers, which we plan to return to later. spielman-srivastava showed how to randomly sample $O(n \log(n)/\epsilon^2)$ edges as to preserve the Laplacian quadratic form of an undirected graph up to an $(1 \pm \epsilon)$-multiplicative factor - which is a stronger approximation guarantee that approximately preserves all cuts as well as many other properties of interest. bss then showed how to sparsify the Laplacian (and preserve all cuts) with only $O(n/\epsilon^2)$ edges deterministically. Removing the log $n$ factor is extremely surprising and consequential. Previously, the log $n$ factor arises for both graph and Laplacian sparsifiers from (essentially) a Chernoff-type union bound. Removing the log $n$ factor required a fundamentally new approach based on carefully studying the eigenvalues associated with a graph. This result has also had many implications outside of computer science including the resolution of the longstanding Kadison-Singer problem from functional analysis [mss].

4.5 Randomized Ford-Fulkerson

Let $G = (V, E)$ be a simple undirected graph, and let $s, t \in V$. We want to compute the maximum $s \rightarrow t$ flow. Recall the Ford-Fulkerson algorithm, a staple of introductory algorithms classes.

Repeatedly find a path from $s$ to $t$ in the residual graph, and route one unit of flow.
Let $\lambda$ denote the value of the max flow. In a simple graph, the $\lambda$ is an integer at most $n$. Thus Ford-Fulkerson makes $n$ iterations, each of which requires $O(m)$ time to find a path. The overall running time is $O(m\lambda) \leq O(mn)$.

Our goal is to improve $O(mn)$. We first consider faster approximations based on the ideas already developed. In the previous section, we showed how to compute a sparse, reweighted subgraph $G'$ of $G$ that preserves all cut values up to a $(1 \pm \epsilon)$-multiplicative factor while. By max-flow min-cut, this also preserves the value of the maximum flow.

**Corollary 4.22.** Let $G = (V, E)$ be a simple undirected graph. For any $\epsilon > 0$, one can compute an $(1 - \epsilon)$-approximation to the value of the maximum flow with high probability in $O\left(m + \frac{n^2 \log^2(n)}{\epsilon^2}\right)$ randomized time.

We now consider the more ambitious challenge of computing the flow exactly in $O(m + n^2 \text{polylog}(n))$ randomized time.

We will analyze a simple algorithm (essentially) due to karger-levine that accelerates Ford-Fulkerson by sampling. Recall that in each iteration, the goal of Ford-Fulkerson is to find just one path from $s$ to $t$. We search the entire graph - size $m$ - to find a single path of $n$ edges. Can we avoid sampling the entire graph?

Consider the first iteration, before we have routed any flow. We know that by applying the random sparsifier in theorem 4.14, we can sample and reweight just $O(n \text{polylog}(n))$ edges while approximating the value of every undirected cut by a constant factor. If every $s \rightarrow t$ cut is approximately preserved, then in particular, we must sample at least one edge from $s$ to $t$. Suppose that for every undirected edge that we sample, we consider any of its directed edges in the bidirected graph. If every $s \rightarrow t$ cut has at least one edge in the sampled set, then we (by max-flow min-cut!) will have a path from $s$ to $t$.

Thus we can randomly sample roughly $n$ edges and find our first path in this smaller set. There is a technical issue in that it still takes that $O(m \text{polylog}(n))$ time to even generate the sample, which will be addressed later.

Consider now the second iteration. We might try the same idea - sample edges based on theorem 4.14 - and look at the corresponding directed edges in the graph. Most of the logic from the first iteration still holds, except for one important exception: some of the originally undirected edges have now become two edges in the same direction. These are precisely the edges used to transport one unit of flow in the first iteration. In the aggregate, all the $s \rightarrow t$ cuts now have 2 less edges than all of the $t \rightarrow s$ cuts! To compensate for the fact that the $s \rightarrow t$ cuts which are a little smaller than before, we might have to sample some extra edges.
4. Sampling edges

4.5. Randomized Ford-Fulkerson

Kent Quanrud
Fall 2022

To further explore the thought experiment, suppose we have now routed half of the maximum flow. Every $s \rightarrow t$ cut loses one edge (in net effect) per unit of flow. Since we have routed only half of the $s \rightarrow t$ maximum flow, and the $s \rightarrow t$ max flow equals the $s \rightarrow t$ min cut, every $s \rightarrow t$ cut has at least half of its edges remaining in the residual graph! When deciding how many edges to sample, then – still in proportion to theorem 4.14 – it is not an unreasonable guess that we would need about twice as many sampled edges to hit all the $s \rightarrow t$ cuts in the residual graph. The main lemma that we will have to prove is the following.

**Lemma 4.23.** Let $\lambda$ be the value of the max flow. Suppose we have routed $(1 - \alpha)\lambda$ units of flow and $\alpha\lambda$ units of flow remain to be routed, where $\alpha > 0$. Let $E' \subset E$ sample (at least) $O\left(\frac{1}{\alpha} n \log^3 n\right)$ edges from the same distribution as in theorem 4.14. Then with high probability, $E'$ contains an $s \rightarrow t$ path.

Suppose first for simplicity that we knew that the maximum flow was $\lambda$. Consider the following algorithm.

In the $i$th iteration, sample $O\left((\lambda n / (\lambda + 1 - i)) \polylog(n)\right)$ undirected edges using the same probabilities as in theorem 4.14, and use the corresponding directed edges in the residual graph when looking for a path from $s$ to $t$ in this sampled set.

Of course we may not actually know the value of the maximum flow. Nonetheless we can effectively guess $\lambda$. We simply repeatedly double the number of sampled edges until we pass the magic threshold required by lemma 4.23 to guarantee an $s \rightarrow t$ path in the residual graph.

Each iteration, sample (directed) edges from the residual graph using the same probabilities as in theorem 4.14, and look for a path from $s$ to $t$. Until a path from $s$ to $t$ is found, keep doubling the number of sampled edges.

Let us first assume lemma 4.23 and prove the running time of $[\text{karger-levine}]$.

**Theorem 4.24 ([karger-levine]).** Let $G = (V, E)$ be a simple undirected graph with $m$ edges and $n$ vertices. Let $s, t \in V$, and let $\lambda \in \mathbb{N}$ be the value of the maximum $s \rightarrow t$ flow. Then one can compute a maximum $s \rightarrow t$ flow in $O((m + n\lambda) \polylog(n))$ randomized time with high probability.

---

2Actually, the algorithm of $[\text{karger-levine}]$ is slightly different, because it samples edges in proportion to sampling probabilities given by $[\text{benczur-karger}]$. The high level ideas are the same, although the log factors are improved by using the $[\text{benczur-karger}]$ sampling probabilities.
Proof. The high-level algorithm is described above, where each iteration we sample edges in proportion to the probabilities from theorem 4.14, until we find an $s \to t$ path. Assuming lemma 4.23 is true, on the $i$th of $\lambda$ iterations, we will find an $s \to t$ path with high probability as soon as we sample at least

$$k_i \overset{\text{def}}{=} O\left(\frac{\lambda}{\lambda - i + 1} n \text{polylog}(n)\right)$$

edges.

We point out that the edge sample probabilities $p_e$ only need to be computed once, at the beginning of the algorithm. Once the $p_e$’s are computed, it is easy to sample one edge at a time in proportion to the $p_e$’s in $O(\log n)$ time per sample.\footnote{Line up the edges in an array and compute the prefix sums of the $p_e$’s. Draw a random number $\theta$ between 0 and $\sum_e p_e$. Binary search for the first edge where the prefix sum up to that edge is at least $\theta$.} Thus generating a sample of $k$ edges takes $O(k \log n)$ time.

The $i$th iteration takes $(k_i \log n)$ time with high probability, because we sample $k_i$ edges and look for paths in these edges. Summed over all $i$, the total amount of time spent on each iteration is

$$O\left(\sum_{i=1}^{n} k_i \log n\right) = O\left(\lambda n \log^{4}(n) \sum_{i=1}^{n} \frac{1}{\lambda + 1 - i}\right) = O\left(\lambda n \log^{5}(n)\right).$$

We also have $m \text{polylog}(n)$ overhead from computing the sampling probabilities $p_e$ initially. This gives the claimed running time. \hfill $\square$

4.5.1 Analysis

Let $f$ be an $s \to t$ flow. Fix an undirected $\{s,t\}$-cut of the form $C = \delta(S)$, where $S \subset V$, $s \in S$, and $t \notin S$. Let $C_f \subseteq C$ be the subset of edges that has a directed edge in the residual graph directed from $s$ to $t$. Our goal is to sample at least one edge out of this subset $C_f$, for every $\{s,t\}$-cut $C$.

For a fixed flow $f$, and an undirected $\{s,t\}$ cut $C = \delta(S)$, let $C_f \subset C$ be the subset of edges that have at least one edge in the residual graph from the $s \to t$ direction.

**Lemma 4.25.** Let $H$ be a set of edges where every edge has connectivity $K$ in the original, undirected graph. Consider all sections of the form

$$A = H_k \cap C_f,$$
where $C = \delta(S)$ is an undirected $\{s, t\}$-cut, and $C_f \subseteq C$ is the subset of undirected edges that have a directed edge in the directed $s \rightarrow t$ cut from $S$ to $V \setminus S$. For all $\alpha \in \mathbb{N}$, there are at most $n^{O(\alpha)}$ distinct sections such that $C$ has at most $\alpha K$ edges.

Proof. In lemma 4.19, we showed that there are at most $n^{2\alpha}$ cuts of the form $A = H \cap C$, where $C$ is an undirected cut with $\leq \alpha K$ edges. Recall the proof of lemma 4.19. To briefly recap, we consider the randomized contraction algorithm except (a) we split off any vertex with degree $< K$ whenever one arises, and (b) we stop when there are $2\alpha$ vertices remaining, and return the set of all undirected cuts induced by subsets of these vertices. Carefully reasoning showed that splitting off was safe, and is in part because splitting off preserves pairwise connectivities and every edge in $H$ has connectivity $\geq K$. By keeping the minimum degree at $K$, we get the same calculations and bounds as when counting the number of $\alpha$-approximate minimum cuts in $\mathbb{C}$.

We modify the above proof very slightly. At the end of the algorithm, when there are $2\alpha$ vertices remaining, we return all $(\leq 2^{2\alpha-2})$ directed $s \rightarrow t$ cuts induced by these vertices. This produces the same calculations and proof (up to a factor of 2) as before when counting approximate minimum cuts (??) and sections of cuts in sparsification (lemma 4.19), hence the same conclusion.\end{proof}

**Lemma 4.26.** Let $k \in \mathbb{N}$ and fix a flow $f$. For an undirected $\{s, t\}$-cut $C = \delta(S)$, where $S \subseteq V$ with $s \in S$ and $t \notin S$, let $C_f \subseteq C$ be the subset of edges that have an edge directed from $S$ to $V \setminus S$ in the residual graph.

Suppose we sample $O\left(\alpha n \log^3(n)\right)$ edges proportional to $p_e$, as defined in section 4.4.2. With probability $1 - 1/\text{poly}(n)$, for all sections of the form $A = H_k \cap C_f$, where $C = \delta(S)$ is an undirected $\{s, t\}$-cut, we have

$$\sum_{e \in A \cap E'} w(e) \geq \frac{3}{4} |A| - \frac{1}{16\alpha \log n} |S|.$$ 

Proof sketch. The claim is very similar lemma 4.20. We only claim the lower bound because that is the only part we require, but an upper bound would follow as well. The proof is similar. We apply Chernoff bounds to each section\footnote{As before, here we are using the following Chernoff bound which previously appeared as an exercise (with $\epsilon = 1/4$): Let $X_1, \ldots, X_n \in [0, 1]$ be independent random variables, and $\epsilon \in (0, 1)$. Then for all $\beta > 0$, $P[X_1 + \cdots + X_n \leq (1 - \epsilon) E[X_1 + \cdots + X_n] - \beta] \leq e^{-\epsilon^2 \beta}.$}, and then use lemma 4.25
to give a refined union-bound where the larger margin of error for larger cuts pays off the fact that there are more sections induced by larger cuts.\footnote{Again, the student reader should verify the calculations for themselves.}

**Lemma 4.27.** Let $\beta > 1$, and suppose we sample $O\left(\beta n \log^3(n)\right)$ edges proportional to $p_e$, as defined in section 4.4.2. With probability $1 - 1/\text{poly}(n)$, for all $\{s,t\}$-cuts, we have

$$
\sum_{e \in C_f \cap E'} w(e) \geq \frac{3}{4}|C_f| - \frac{1}{\beta} |C|.
$$

**4.5.2 The final touch**

**Lemma 4.28.** Suppose the residual graph has maximum $\{s,t\}$-flow $\alpha \lambda$, where $\lambda$ is the value of the maximum flow originally. For every undirected $\{s,t\}$-cut $C = \delta(S)$, where $s \in S \subseteq V - t$,

$$|C_f| \geq \alpha |C|/2.$$

**Proof.** Let $D$ denote the directed $s \to t$ cut induced by $S$. By max-flow min-cut, we have $|D| \geq \lambda$ initially, and each unit of flow decreases $|D|$ by at most 1. After routing $(1 - \alpha)\lambda$ units of flow, we have $|D| \geq (1 - \alpha)\lambda$. We also have $|C_f| \geq |D|/2$ because each (undirected) edge $e \in C_f$ contributes at most 2 directed edges to $D$. Together we obtain $|C_f| \geq \alpha |C|/2$. \hfill \Box

Now we can prove lemma 4.23.

**Lemma 4.23.** Let $\lambda$ be the value of the max flow. Suppose we have routed $(1 - \alpha)\lambda$ units of flow and $\alpha \lambda$ units of flow remain to be routed, where $\alpha > 0$. Let $E' \subset E$ sample (at least) $O\left(\frac{1}{\alpha} n \log^3 n\right)$ edges from the same distribution as in theorem 4.14. Then with high probability, $E'$ contains an $s \to t$ path.

**Proof.** For all $\{s,t\}$-cuts $C$, we have

$$
\sum_{e \in C_f \cap E'} w(e) \geq \frac{3}{4}|C_f| - \frac{\alpha}{4}|C| \overset{(b)}{=} 0
$$

where (a) is by lemma 4.27 and (b) is by lemma 4.28. Thus we sample at least one edge from every directed $\{s,t\}$-cut in the residual graph, which implies that there is an $s \to t$ path. \hfill \Box
4.6 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.

4.7 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.

1. Briefly describe how to modify the random-contractions to return a $k$-cut.

2. Analyze the probability that your modified algorithm returns a minimum $k$-cut.8

3. Describe and analyze an algorithm, using your modified random-contractions as a subroutine, that computes a minimum $k$-cut with high probability in $O\left(n^{c_1k \log c_2 n}\right)$ 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.)

---

7 An earlier version of this note had an exercise about graphs with large minimum degree, which I realized was wrong. (In the previous version, this was exercise 3.)

8 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 4.3.** Consider the minimum cut problem in undirected graphs. For $\alpha \geq 1$, we say that a cut $C = \delta(S)$ is a 2-approximate minimum cut if its capacity is at most twice the capacity of the minimum cut.

1. Let $C$ be an 2-approximate minimum cut. Prove that the random-contractions algorithm returns $C$ with probability $\geq 1/\binom{n}{\lceil \frac{n}{4} \rceil}$.

2. Show the number of 2-approximate minimum cuts is at most $\binom{n}{4}$.

**Exercise 4.4.** Let $G = (V, E)$ be a simple Eulerian directed graph.

1. Design and analyze an algorithm that computes a reweighted subgraph $(G' = (V, E'), w : E' \to \mathbb{R}_{>0})$ with $O(n \text{ poly}(\log n, 1/\epsilon))$ edges that preserves the value of every directed cut up to an $(1 \pm \epsilon)$-multiplicative factor.

2. Design an analyze a $O((m + n^2) \text{ polylog}(n))$ time algorithm for maximum flow on simple Eulerian graphs.

3. Suppose you were given a simple directed graph $G$ that was “approximately Eulerian” up to an $\alpha$-multiplicative factor. (For example, $\alpha = 2$.) That is, suppose that the in-degree was always within a multiplicative factor of the out-degree. Can one design sparsifiers and faster max flow algorithms for such graphs? If no, what goes wrong? If yes, what is the dependency on $\alpha$? Here a relatively brief answer will suffice.

---

*A directed graph with Eulerian if every vertex has the same in-degree as out-degree*
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 $\frac{1}{2^{100}}$. 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. Even though the event of getting any one of these counts should be low, being far from average, 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 $2.818 \times 10^{-7}$.

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 $1.364 \times 10^{-10}$.
We generalize the discussion to coins with any fixed probability of heads, \( p \in [0, 1] \). The binomial distribution, denoted \( B(n, p) \), 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 [wikipedia-binomial-distribution].)

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

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

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

**Proof.** We have

\[
P[B \leq (1 - \epsilon)pn] = P[e^{-\epsilon B} \geq e^{-\epsilon(1-\epsilon)pn}] \leq \frac{E[e^{-\epsilon B}]}{e^{-\epsilon(1-\epsilon)pn}} = e^{\epsilon(1-\epsilon)pn} E[e^{-\epsilon B}]
\]

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

\[
E[e^{-\epsilon B}] = E[e^{-\epsilon X_1 - \epsilon x_2 - \cdots - \epsilon X_n}] \leq E[e^{-\epsilon X_1}] E[e^{-\epsilon X_2}] \cdots E[e^{-\epsilon X_n}],
\]

where (b) is by independent of the \( X_i \)'s. For each \( X_i \), we have

\[
E[e^{-\epsilon X_i}] = pe^{-\epsilon} + (1 - p) = 1 + p(e^{-\epsilon - 1}) \leq (1 - \epsilon + \epsilon^2/2) + (1 - p) = 1 - (\epsilon - \epsilon^2/2)p \leq e^{-(\epsilon - \epsilon^2/2)p}.
\]
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,

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

Putting everything together, we have

$$P[B \leq (1 - \epsilon)p n] \leq e^{\epsilon(1-\epsilon)p n} E[e^{-\epsilon B}] \leq e^{\epsilon(1-\epsilon)p n - (\epsilon^2/2)p n} = e^{-\epsilon^2 p n/2},$$

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 5.2.

**Lemma 5.2.** Let $B \sim B(n, p)$ and $\epsilon \in (0, 1)$. Then

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

### 5.2 Random graphs

Paul Erdös, inspired by *ramsey* before him, had a series of work analyzing *random graphs*, producing a large body of results that can mostly be grouped into two

---

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 $G(n, .01)$ [wikipedia-erdos-renyi-jpg].
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 properties. Today we will study the $G(n,p)$ random graph, sometimes called Erdős-Rényi graphs based on work by *erdos-renyi-1959; erdos-renyi-1960*. A random graph from $G(n,p)$ is an undirected graph over $n$ vertices, where every edge is sampled independently with probability $p$. By now there is a large 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, Erdős 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 $G \sim G(n,p)$ 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 $G$ has 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 a line of research has obtained a much more refined and detailed understanding than stated in theorem 5.3. We refer the reader to [bollobas] 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 5.1.

### 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, like 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 $G(n, p)$, with probability of error $\leq 1/n^2$, no component has $k$ vertices for any value $k$ in the interval

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

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. In conjunction with the gap theorem (lemma 5.4), which rules out all intermediate sizes, this implies 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 $G(n, p)$ 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. That is, we almost certainly connect $C_1$ and $C_2$. Since there is a limited number of 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 $c > 1$.

5.2.2 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-90 in the following delightfully simple way.

**Theorem 5.6.** 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-90 (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 \( B(n, p) \), arc \((u, v)\) 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.

5.3 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-90. His proof is also described in [bollobas]. We will also reuse some of the ideas in the proof to analyze the \( p < 1/n \) in ???. 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 \( G(n, p) \), with probability of error \( \leq 1/n^2 \), no component has \( k \) vertices for any value \( k \) in the interval

\[
\frac{C\log(n)}{\epsilon^2} \leq k \leq \frac{en}{C}.
\]

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
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 $i$th 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 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 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 $i$th 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 $C(v) = B_i = A_i$.

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 $|A_i| \neq i$, $A_{i+1}$ 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 $v_i$. 

97
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 $x \not\in A_i$ 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., $|A_i| \sim \mathcal{B}(n - 1, 1 - (1 - p)^i)$.

**Lemma 5.7.** Let $i \leq \epsilon n / 2((1 + \epsilon))$. Then $E[|A_i|] \geq (1 + \epsilon)i$.

**Proof.** We have

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

where (a) is because $ip = (1 + \epsilon)i/n \leq \epsilon/4$. Thus

$$E[|A_i|] = 1 + (1 - (1 - p)^i)(n - 1) \geq (1 - \epsilon/4)ipn \geq (1 + \epsilon)i.$$

**Lemma 5.8.** Let $i \leq \epsilon n / 2(1 + \epsilon)$. Then $P[|A_i| \leq i] \leq e^{-\epsilon^2i/8}$.

**Proof.** We have

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

Here (a) is by lemma 5.7. (b) is by the tail inequality on binomial distributions, lemma 5.1.

Let $I = \{i \in \mathbb{N} : 32 \ln(n)/\epsilon^2 \leq i \leq \epsilon n / 2((1 + \epsilon))\}$. For all $i \in I$, we have

$$P[|A_i| \leq i] \leq 1/n^4.$$ 

By the union bound, we have

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

Thus with probability $\geq 1 - 1/n$, 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. Random Sums and Graphs

5.4 Galton-Watson branching processes for general branch factor

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 $3 \leq h \leq \epsilon n$, 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 5.9.** 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

\[ p_0 = 1, \]
\[ p_1 = 1 - (1 - 1/k)^k \geq 1 - e^{-1} \geq .63, \]
\[ p_2 = 1 - (1 - .63/k)^k \geq 1 - e^{-63} \geq .467, \]
\[ p_3 \geq 1 - (1 - .467/k)^k \geq 1 - e^{-467} \geq .373 \geq 1/3. \]

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,

\[ p_{i+1} \geq 1 - (1 - 1/ik)^k \geq 1 - e^{-1/i} \geq \frac{1}{i} - \frac{1}{2i^2} \geq \frac{1}{i+1}. \]

Here (a) is by induction. (b) 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 \( \leq h \), for any \( h \leq cn/(1+\epsilon) \). Recall the sets \( A_0, A_1, A_2, \ldots \) from section 5.3. Given that \( |A_i| \leq h \), we can think of \( A_{i+1} - A_i \) as adding (at least) \( n/(1+\epsilon) \) children each with probability \( p = (1+\epsilon)/n \). Either we find new elements for all \( h \) rounds - which forces \( |A_h| \geq h \) - or we hit \( |A_i| = h \) at some point \( i < h \). Thus the odds of \( v \) acquiring \( h \) vertices in its connected component is at least the odds produced by theorem 5.9 for this value of \( h \); namely, \( 1/h \). This gives us lemma 5.5.

### 5.5 Exercises

**Exercise 5.1.** Prove item 2 of theorem 5.3.\(^2\)

**Exercise 5.2.** Prove lemma 5.2. (Here the important part is not the constant, \( 1/3 \) – any constant \( c > 0 \) is already interesting.)

\(^2\)It may be helpful to understand the proof of the gap theorem (lemma 5.4) in section 5.3.
5.6 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.
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 problems in this category is integer programs (IPs), which are mathematical optimization problems where the variables are required to be integers.

One example is a covering integer program, which is an optimization problem of the form

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n} c_j x_j \\
\text{over} & \quad x \in \mathbb{Z}_{\geq 0} \\
\text{s.t.} & \quad \sum_{j=1}^{n} A_{ij} x_j \geq b_i \text{ for } i \in [m],
\end{align*}
\]

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 \( b_i. \)

The special case where \( c = 1, b = 1, \) and \( A \in \{0,1\}^{m \times n} \) describes the set cover problem. There 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

\[
\text{maximize } \sum_{j=1}^{n} b_j y_j \\
\text{over } y \in \mathbb{Z}_{\geq 0}^n \\
\text{s.t. } \sum_{j=1}^{m} A_{ij} y_j \leq c_i \text{ for } i \in [m]
\]

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.

\[
\text{minimize } \sum_{j=1}^{n} c_j x_j \\
\text{over } x \in \mathbb{R}_{\geq 0}^n \\
\text{s.t. } \sum_{j=1}^{n} A_{ij} x_j \geq b_i \text{ for } i \in [m].
\]

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.

Unlike integer programs, linear programs are polynomial time solvable (!). This allows for the following general approach to discrete optimization: given an IP

---

\(^1\)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.
6. Randomized Rounding

6.1. SAT

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., [WilliamsonShmoys; Vazirani01]). 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 \{t, 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 \( k \).

<table>
<thead>
<tr>
<th>( k )</th>
<th>oblivious APX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1/2)</td>
</tr>
<tr>
<td>2</td>
<td>(3/4)</td>
</tr>
<tr>
<td>3</td>
<td>(7/8)</td>
</tr>
<tr>
<td>4</td>
<td>(15/16)</td>
</tr>
<tr>
<td>5</td>
<td>(31/32)</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( k )</td>
<td>(1 - 1/2^k)</td>
</tr>
</tbody>
</table>

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 \( f(x_1, \ldots, x_n) \) of \( n \) variables, consisting of \( m \) clauses \( C_1, \ldots, C_m \). Let \( \text{OPT} \) 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 “\( \overline{x}_j \in C_i \)” indicate that \( \overline{x}_j \) appears in \( C_i \).

Consider now the following integer program.

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{m} z_i \\
\text{subject to} & \quad \sum_{j : x_j \in C_i} y_j + \sum_{j : \overline{x}_j \in C_i} (1 - y_j) \geq z_i \quad \text{for all clauses } C_i.
\end{align*}
\]

The integer program has \( \{0, 1\} \)-variables for each variable \( x_j \) and each clause \( C_i \), with the following interpretations.

1. For \( j = 1, \ldots, n \), let \( y_j \in \{0, 1\} \) indicates whether we set \( x_j = t \) \((y_j = 1)\) or \( x_j = f \) \((y_j = 0)\).

2. For \( i = 1, \ldots, m \), let \( z_i \in \{0, 1\} \) indicate whether we satisfy the \( i \)th 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 \( \overline{x}_j \in C_i \).

**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 \([0, 1]\).

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{m} z_i \\
\text{subject to} & \quad \sum_{j : x_j \in C_i} y_j + \sum_{j : \overline{x}_j \in C_i} (1 - y_j) \geq z_i \quad \text{for all clauses } C_i \\
& \quad 0 \leq z_i \leq 1 \quad \text{for all } i = 1, \ldots, m \\
& \quad 0 \leq y_j \leq 1 \quad \text{for all } j = 1, \ldots, n
\end{align*}
\]

Let \( \text{OPT}_{lp} \) denote the optimum value of the LP. Since the LP is a relaxation of the IP above it, we have \( \text{OPT}_{lp} \geq \text{OPT} \).
Rounding the LP solution. Let $y_1, \ldots, y_n$ and $z_1, \ldots, z_m$ be an optimum solution to the LP. We know that the objective value $\text{OPT}_{\text{LP}} = \sum z_i$ is very good. Our goal now is to convert the $y_j$’s into discrete decisions while keeping the objective value as close to $\text{OPT}_{\text{LP}}$ 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 $x_1 = t$ half the time, and $x_1 = f$ the other half. Consider the following randomized rounding algorithm:

**Randomized rounding for SAT**

1. Let $y_1, \ldots, y_n$ be an optimum solution to the LP for max-SAT.

2. For each variable $x_j$, independently, randomly set $x_j = t$ with probability $y_j$, and $x_j = f$ 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 $x_j = f$ for all $x_j \in C_i$, and set $x_j = t$ for all $\bar{x}_j \in C_i$. Thus

$$
P[C_i \text{ not satisfied}] = \prod_{j: x_j \in C_i} (1 - y_j) \prod_{j: \bar{x}_j \in C_i} y_j.
$$

Now, by the inequality $1 + x \leq e^x$ (for all $x$), we can simply the RHS as

$$
e^{-\sum_{j: x_j \in C_i} y_j + \sum_{j: \bar{x}_j \in C_i} (1 - y_j)} \leq e^{-z_i}.
$$

Here the inequality follows from the LP constraint for $C_i$. Finally, by convexity$^2$ of $f(x) = e^{-x}$, we have

$$
e^{-z_i} \leq (1 - z_i)e^0 + z_ie^{-1} = 1 - (1 - 1/e)z_i,
$$

as desired. $\square$

---

$^2$f(x) is convex if $f(ta + (1 - t)b) \leq tf(a) + (1 - t)f(b)$ for all $a, b$ and all $t \in [0, 1]$. 

106
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 \((1 - 1/e)\) OPT_{LP} \geq (1 - 1/e) 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 \(C_i = x_1 \lor x_2\). (It will be obvious how to generalize the analysis to other pairs of variables.) We have

\[
P[C_i \text{ not satisfied}] = (1 - y_1)(1 - y_2) \leq \left(\frac{(1 - y_1) + (1 - y_2)}{2}\right)^2 \\
\leq \left(1 - \frac{z_j}{2}\right)^2 \leq (1 - z_j) \left(1 - \frac{0}{2}\right)^2 + z_j \left(1 - \frac{1}{2}\right)^2 \\
= 1 - z_j + \frac{z_j}{4} = 1 - \frac{3}{4}z_j.
\]

(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.

<table>
<thead>
<tr>
<th>(k)</th>
<th>oblivious</th>
<th>LP</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>(z_i)</td>
<td>(\geq (3/4)z_i)</td>
</tr>
<tr>
<td>2</td>
<td>3/4</td>
<td>((3/4)z_i)</td>
<td>(\geq (3/4)z_i)</td>
</tr>
<tr>
<td>3</td>
<td>7/8</td>
<td>((1 - 1/e)z_i)</td>
<td>(\geq (3/4)z_i)</td>
</tr>
<tr>
<td>4</td>
<td>15/16</td>
<td>((1 - 1/e)z_i)</td>
<td>(\geq (3/4)z_i)</td>
</tr>
<tr>
<td>5</td>
<td>31/32</td>
<td>((1 - 1/e)z_i)</td>
<td>(\geq (3/4)z_i)</td>
</tr>
<tr>
<td></td>
<td>(\vdots)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k)</td>
<td>(1 - 1/2^k)</td>
<td>((1 - 1/e)z_i)</td>
<td>(\geq (3/4)z_i)</td>
</tr>
</tbody>
</table>
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 6.1.

### 6.2 Set cover

In *set cover*, we are given $m$ points $[m] = \{1, \ldots, m\}$, and $n$ sets $S_1, \ldots, S_n \subseteq [m]$. The goal is to

*find the minimum number of sets $S_{i_1}, \ldots, S_{i_k}$ 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.

$$\text{minimize } \sum_{j=1}^{n} x_j \text{ over } x \in \mathbb{R}_{\geq 0}^n \text{ s.t. } \sum_{S_j \ni i} x_j \geq 1 \text{ for all } i \in [m].$$

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 \( \text{OPT} \)?
2. Is \( F \) a set cover?

For the first question, we have

\[
E[|F|] = \sum_j P[S_j \in F] = \sum_j x_j = \text{OPT}_{\text{LP}},
\]

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

\[
E[\# \text{ sets in } F \text{ covering } i] = \sum_{S_j \ni i} P[S_j \in F] = \sum_{S_j \ni i} x_j \geq 1.
\]

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

\[
P[F \text{ doesn’t cover } i] = \prod_{S_j \ni i} (1 - x_j) \leq e^{-\sum_{S_j \ni i} x_j} \leq e^{-1}.
\]

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

$$P[F \text{ doesn’t cover } i] = \prod_{S_j \ni i} (1 - \alpha x_j) \leq e^{-\sum_{S_j \ni i} \alpha x_j} \leq e^{-\alpha} = 1/m^2. \quad (6.2)$$

**Lemma 6.6.** $F$ is a set cover with probability at least $1 - 1/m$.

**Proof.** By the union bound, we have

$$P[F \text{ is not a set cover}] \leq \sum_i P[F \text{ doesn’t cover } i] \leq \frac{1}{m},$$

as desired.

**Theorem 6.7.** Randomized rounding is a $O(\log m)$-approximation algorithm for set cover.

**Proof.** We have $E[|F|] = 2 \log(m) \text{OPT}$, and by Markov’s inequality, $|F| \leq 4 \log(m) \text{OPT}$ 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 $4 \log(m) \text{OPT}$.

**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 \( c \in \mathbb{R}_{>0}^n, A \in \mathbb{R}_{\geq 0}^{m \times n}, \) and \( b \in \mathbb{R}_{>0}^m. \) The matrix notation expands out to the following.

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n} c_j x_j \\
\text{over} \quad & x \in \mathbb{Z}_{\geq 0}^n \\
\text{s.t.} \quad & \sum_{j=1}^{n} A_{ij} x_j \geq b_i \text{ for } i \in [m].
\end{align*}
\]

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 an \( O(\log m) \) 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.

**Theorem 6.8** (Multiplicative Chernoff bounds). Let \( X_1, \ldots, X_n \in [0,1] \) be independent random variables, and \( \epsilon \in [0,1]. \) Let \( \mu = \mathbb{E}[X_1 + \cdots + X_n]. \) Then

\[
\begin{align*}
\mathbb{P}[X_1 + \cdots + X_n \geq (1 + \epsilon)\mu] & \leq e^{-\epsilon^2 \mu/3}, \\
\mathbb{P}[X_1 + \cdots + X_n \leq (1 - \epsilon)\mu] & \leq e^{-\epsilon^2 \mu/2}.
\end{align*}
\]

The main part of the analysis is to show that a single constraint is satisfied with high probability, as follows.

**Lemma 6.9.** The \( i \)th constraint is satisfied with probability at least \( 1 - 1/m^2. \)
6. Randomized Rounding

6.3. Covering integer programs

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 \( \mu = E \left[ \sum_{j=1}^n X_j \right] \).

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

\[
\mu \geq \alpha \left( 1 - \sum_{j=1}^n A_{ij} \lfloor \alpha x_j \rfloor \right), \quad \text{and} \quad \mu \geq \alpha - 1 \geq 8 \log(m).
\]

We have

\[
P \left[ z \text{ fails constraint } i \right] = P \left[ \sum_{j=1}^n A_{ij} z_j \leq 1 \right] = P \left[ \sum_{j=1}^n X_j \leq 1 - \sum_{j=1}^n A_{ij} \lfloor \alpha x_j \rfloor \right] \leq P \left[ \sum_{j=1}^n X_j \leq \frac{\mu}{\alpha} \right].
\]

By the Chernoff inequality, we have

\[
P \left[ \sum_{j=1}^n X_j \leq \frac{\mu}{\alpha} \right] \leq e^{- \left( 1 - 1/\alpha \right)^2 \mu / 2}.
\]

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

\[
P \left[ z \text{ fails constraint } i \right] \leq e^{- \left( 1 - 1/\alpha \right)^2 \mu / 2} \leq e^{-2 \ln(m)} = \frac{1}{m^2},
\]

as desired.

\[
\]

Theorem 6.10. With constant probability, the randomized rounding algorithm returns a feasible solution of cost at most \( O(\ln(m)) \text{OPT}_{LP} \).
6. Randomized Rounding

6.4. Exercises

Proof. By Markov’s inequality, we have
\[ \sum_j c_j z_j \leq 2 \mathbb{E} \left[ \sum_j c_j z_j \right] = 2 \alpha \text{OPT}_{LP} \]
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 \( 2 \text{OPT}_{LP} \) is at most
\[ \frac{1}{2} + \mathbb{P}[z \text{ fails any constraint}] \leq \frac{1}{2} + \sum_i \mathbb{P}[z \text{ fails constraint } i] \leq \frac{1}{2} + \frac{1}{m}, \]
as desired. \( \square \)

6.4 Exercises


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 \( S_1, \ldots, S_n \subseteq [m] \) and costs \( c_i > 0 \) for each set \( S_i \). The goal is to compute the minimum cost collection of sets covering \([m] \). We say 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 \textit{round-and-fix} for which some pseudocode is given below. \textit{round-and-fix} is similar to randomized rounding and has two stages. The first stage solves the LP and then \textit{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 \textit{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.
**6. Randomized Rounding**

**6.4. Exercises**

Kent Quanrud  
Fall 2022

**round-and-fix**(*sets* $S_1, \ldots, S_n \subseteq [m]$, *costs* $c \in \mathbb{R}_{>0}^n$, $\alpha \geq 1$)

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 $\alpha$, 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.

**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.

**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.
Chapter 7

Distinct elements

7.1 Unique visitors

Let’s say you have a website. You might like to know how many people visit your website. You set up simple counter on your server to keep track of how many HTTP requests you serve. You are pleased to discover that this counter increases steadily — apparently there are many visits to your website. Upon investigation, however, you find out that almost all of these visits are from the same bot, canvassing your web page for who-only-knows-what reason. In fact, what you are really interested in is not the number of visits of your webpage, but the number of distinct visitors to your website. Counting the number of distinct visitors might be the most popular feature of Google Analytics and other similar software analyzing web traffic.

How can we count the number of distinct visitors? Suppose each visitor has a unique identifier, such as an IP address. We can store the set of different visitors in a dictionary, maybe even using the linear probing hash table previously discussed. Even the best dictionaries, however, will ultimately require space proportional to the number of keys. For your website, this could be the total number of IP addresses in the world, and only so many IP addresses can fit in RAM. In fact, any exact implementation will always require a lot of space (see exercise 7.1.)

Google Analytics has a help page where they explain how their unique visitors count works (https://support.google.com/analytics/answer/2992042). There is an interesting paragraph where they note that in 2017 they switched to a new algorithm to “more efficiently count users with high accuracy and low error rate (typically less than 2%)”. From this we can extract some interesting observations. First, they are not reporting an exact count, but rather one with a “low error rate”. That is, they are approximating. Second, they do not guarantee any particular rate of error. It is “typically” less than 2%, but apparently the error rate can vary and is not always

\[1\] In reality, this may be a combination of IP address and cookies, partly because the same IP address can serve many people from the same network.
less than 2%. Lastly, they did something in early 2017 to improve their algorithm, and “more efficiently count users”. This suggests that distinct elements is an active problem where having a better algorithm still matters\(^2\). A little more searching shows a 2014 publication out of Google about distinct elements [hnh], that explains some low-level enhancements to a well-known algorithm for the distinct elements problem called HyperLogLog [ffgm].

There are many natural applications for distinct elements besides internet-scale streaming. In databases, quick estimates of the number of distinct elements are used to optimize complex queries [hnss]. Having a crude estimate of the number of distinct elements can be useful in deciding what kind of data structure or algorithmic strategy to pursue when processing these elements. If the count is very small, then maybe an asymptotically poor approach with very good constants is actually faster. If the count is very large, then asymptotics kick in and one should go for the asymptotically optimal algorithms, even if the implementation is clumsy and the constants are bigger.

We briefly review the streaming model. We have elements coming in one at a time from a stream, from the set of integers \([n] = \{1, \ldots, n\}\). Let \(m\) denote the total number of elements in the stream. Items may repeat. Our goal is to compute the distinct number of items in the stream. We let \(k\) to denote the number of distinct items.

We mention in passing that the number of distinct elements can be interpreted as the \(L_0\)-norm of the frequency vector. Recall that the frequency vector is the vector with one coordinate per item counting the frequency of that item. Other norms, such as \(L_2\) and \(L_p\), and other quantities such as entropy, are helpful for understanding the “shape” of the data. We refer the reader to [ams] for more on \(L_p\)-frequency estimation. Algorithms for these problems have found further use as extremely efficient data structures inside fast approximation algorithms, particularly for accelerating primitives from linear algebra.

**First ideas.** How do we begin to design an efficient approximation algorithm for the distinct elements problem? The reader might guess based on previous discussions that hashing will be useful. Indeed, we will be using hashing. Let us take as a starting point the count-min data structure for the heavy hitters problem (chapter 2). A salient point about count-min is that some internal state changes with every element that is processed. On a given item \(e\), count-min hashes \(e\) several times and increase a number of counters. Consider now the distinct elements problem. In this setting, the same item \(e\) can reappear again and again and again and the number of distinct elements does not change. This is a bad sign for count-min, or any other algorithm

\(^2\)Of course, there may be other details of the real-world problem, beyond the clean mathematical version we will discuss, that Google may have improved.
that constantly updates its state. One prefers an approach that is impervious to repeating elements.

As a first hint towards a new approach, suppose the reader was given access to an ideal hash function $h : [n] \to [0, 1]$. That is, for each item $e$, we independently sample a unique and continuous value $h(e) \in [0, 1]$. (Later, after developing the main algorithmic ideas, we will return to this assumption on $h$ and replace it with a more realistic alternative.) Does an ideal and continuous function such as $h$ inspire any ideas?

As a second hint, suppose the reader was presented with the following statistical facts. How might the following lemma be employed in the service of estimating the number of distinct elements?

**Lemma 7.1.** Let $Y_1, \ldots, Y_k \in [0, 1]$ be independent and distributed uniformly at random. Let $X = \min\{Y_1, \ldots, Y_k\}$.

1. $E[X] = \frac{1}{k+1}$.
2. $E[X^2] = \frac{2}{(k+1)(k+2)}$.

The proof of lemma 7.1 is left as exercise 7.3.

### 7.2 Preliminaries on continuous random variables

We stated lemma 7.1 before properly introducing continuous random variables. As with discrete random variables, we assume the reader has had some familiarity with continuous random variables. Here we briefly review the basic notions, which should confirm the reader's common sense.

Consider a uniformly random variable $X \in [0, 1]$. That is, $X$ has “equal probability” of being any particular value in $[0, 1]$. Now, the “probabilities” associated with $X$ are a little less straightforward than with finite and discrete random variables, but the reader will likely find it to still be totally natural.

For any fixed $\alpha \in [0, 1]$, say $\alpha = .511989$, the probability that $X$ is exactly $\alpha$ is $0$. The fact that a continuous random variable has “zero probability” at every point may seem odd but it is not problematic. It is completely analogous to a line segment having no area in two dimensions, or a planar surface having no volume in three dimensions. For a continuous random variable such as $X$, the probability at a specific point is the wrong question to ask. Instead, let consider any two fixed values $a, b \in [0, 1]$ with $0 \leq a \leq b \leq 1$. Then we have

$$
P[a \leq X \leq b] = b - a.
$$
This is totally intuitive. We note that from the formal perspective of measure theory, understanding the probability of $X$ lying in any particular (open or closed) interval suffices to define $X$. (See, e.g., billingsley.)

Any continuous random variable $X \in \mathbb{R}$ we consider will implicitly be equipped with a density function $f : \mathbb{R} \to [0, 1]$. Then the probability of $X$ lying in an interval $[a, b]$ is given by

$$\mathbb{P}[a \leq X \leq b] = \int_{a}^{b} f(t) \, dt.$$  

We are implicitly assuming that $f$ is integrable. The expected value of a continuous random variable $X \in \mathbb{R}$ with density function $f(t)$ is given by the following.

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} t f(t) \, dt.$$  

This is the continuous analogue of the definition of expected value for discrete random variables, with the (discrete) sum replaced by a (continuous) integral.$^3$

We note that Markov’s inequality applies to continuous random variables as well. The argument is the same.

**Lemma 7.2.** Let $X \geq 0$ be a nonnegative random variable (discrete or continuous). Then for any $\alpha \geq 1$,

$$\mathbb{P}[X \geq \alpha \mathbb{E}[X]] \leq \frac{1}{\alpha}.$$  

### 7.3 Distinct elements, continued

Let us return to our discussion on developing an (idealized) algorithm for counting the distinct elements. The reader was presented with the following statistical facts about the minimum of independent uniform continuous random variable. How can one employ the following statistical facts algorithmically?

**Lemma 7.1.** Let $Y_1, \ldots, Y_k \in [0, 1]$ be independent and distributed uniformly at random. Let $X = \min\{Y_1, \ldots, Y_k\}$.

1. $\mathbb{E}[X] = \frac{1}{k+1}$.

$^3$In measure theory, discrete and continuous random variables are unified under the notion of measurable random variables. For finite variables, the summation over its values is interpreted as an integral over an appropriate discrete topology. See billingsley.
2. \( \mathbb{E}[X^2] = \frac{2}{(k+1)(k+2)}. \)

One can use the idealized, continuous hash function \( h : [m] \rightarrow [0, 1] \) to assign to each distinct element an independent and uniformly sampled value in \([0, 1]\). Consider the minimum hash value seen in the stream. The set of hash values seen, over \( k \) distinct elements, is precisely a set of \( k \) independent \([0, 1]\)-random variables. We make three basic observations.

1. The minimum hash does not change in the face of duplicate elements. It is strictly a function of the set of distinct elements that have appeared in the stream.

2. We only have to keep track of one number, which is space friendly.

3. By part 1 of lemma 7.1, there is an explicit connection between the minimum hash and the number of distinct elements.

We expand on the third observation. If there are \( k \) distinct elements, generating \( k \) uniformly random numbers \( Y_1, \ldots, Y_k \in [0, 1] \), then the minimum, \( X = \min\{Y_1, \ldots, Y_k\} \), has expected value

\[
\mathbb{E}[X] = \frac{1}{k+1},
\]

by lemma 7.1. This suggests using \( (1/X) - 1 \) as an estimator for \( k \). However, we have not shown that \( \mathbb{E}[(1/X) - 1] \) equals \( k \), nor are we going to. Instead, our analysis is based on showing that \( X \) is close enough to \( 1/(k+1) \) to guarantee that \( (1/X) - 1 \) is closed to \( k \).

**Lemma 7.3.** Let \( \epsilon > 0 \) be sufficiently small. If \( \frac{1-\epsilon}{k+1} \leq X \leq \frac{1+\epsilon}{k+1} \), then

\[
(1-3\epsilon)k \leq \frac{1}{X} - 1 \leq (1+3\epsilon)k.
\]

**Proof.** We have the following equivalent inequalities:

\[
\frac{1-\epsilon}{k+1} \leq X \leq \frac{1+\epsilon}{k+1} \iff \frac{k+1}{1+\epsilon} \leq \frac{1}{X} \leq \frac{k+1}{1-\epsilon} \iff \frac{k-\epsilon}{1+\epsilon} \leq \frac{1}{X} - 1 \leq \frac{k+\epsilon}{1-\epsilon}.
\]

In the last equation, we observe that the LHS is \( \geq (1-3\epsilon)k \) and the RHS is \( \leq (1+3\epsilon)k \) for \( k \geq 1 \). \( \square \)
Recall that $E[X] = 1/(k+1)$. If one could show that

$$|X - E[X]| \leq \frac{\epsilon}{k+1}$$

with high probability, then $1/X - 1$ would give an accurate estimate from $k$ with high probability. Unfortunately, $\epsilon/(k+1)$ is a very small margin of error. For example, suppose we tried to apply Markov’s inequality. Markov’s inequality implies that

$$P(|X - E[X]| \geq \epsilon/(k+1)) \leq \frac{E[|X - E[X]|]}{(\epsilon/(k+1))^2}.$$  \hspace{1cm} (7.1)

The RHS is challenging for two reasons. First, there is a potentially large factor $(k+1)/\epsilon$, so $E[|X - E[X]|]$ needs to be much smaller then $\epsilon/(k+1)$ to obtain interesting probabilities. The second issue is that the expectation of the absolute value, $E[|X - E[X]|]$, is hard to analyze. Consider instead the following analysis that at least addresses the second issue by squaring both sides before applying Markov’s inequality. We have

$$P(|X - E[X]| \geq \frac{\epsilon}{k+1}) = P((X - E[X])^2 \geq \frac{\epsilon^2}{(k+1)^2}) \leq \frac{E[(X - E[X])^2]}{\epsilon^2/(k+1)^2}.$$  \hspace{1cm} (7.1)

Consider the numerator. We have

$$E[(X - E[X])^2] \equiv E[X^2] - 2XE[X] + E[E[X]^2] = E[X^2] - E[X]^2$$ \hspace{1cm} (7.2)

where (a) applies linearity of expectation. By our given statistics on the minimum of $k$ uniform random variables (lemma 7.1), we have

$$E[X^2] - E[X]^2 = \frac{2}{(k+1)(k+2)} - \frac{1}{(k+1)^2} = \frac{k}{(k+1)^2(k+2)}.$$

Plugging back in, we have

$$P(|X - E[X]| \geq \frac{\epsilon}{k+1}) \leq \frac{E[(X - E[X])^2]}{\epsilon^2/(k+1)^2} = \frac{k}{\epsilon^2(k+2)}.$$

This bound is much cleaner than the earlier one in (7.1). Alas, $\frac{k}{\epsilon^2(k+2)}$ is bigger than 1 for any interesting $\epsilon$ and $k$, so the bound we obtained is not very useful.
7.4 Variance and Chebyshev’s inequality

While our analysis above may not have totally worked out, it used some interesting tools that are worth pointing out.

The variance of a random variable $X$, sometimes denoted $\text{Var}[X]$, is the quantity

$$\text{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

We always have the identity,

$$\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2,$$

which appears in (7.2).

**Lemma 7.4 (Chebyshev’s inequality).** Let $X \in \mathbb{R}$ be a random variable. For any $\alpha > 0$,

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq \alpha] \leq \frac{\text{Var}[X]}{\alpha^2}.$$

In the above analysis, we implicitly proved and then applied Chebyshev’s inequality for $\alpha = \epsilon/(k+1)$. (The proof, to recap, is to square both sides and then apply Markov’s inequality.) Chebyhsev’s inequality, combined with our analysis of the variance of $X$, gave us

$$\mathbb{P}\left[|X - \mathbb{E}[X]| \geq \frac{\epsilon}{k + 1}\right] \leq \left(\frac{k + 1}{\epsilon}\right)^2 \text{Var}[X] = \frac{k}{\epsilon^2(k + 2)}.$$

Our attempt to analyze $X = \min\{Y_1, \ldots, Y_k\}$ above fell short because $\text{Var}[X]$ was too large. How can we reduce $\text{Var}[X]$?

7.5 Amplification: variance reduction and the median trick

How do we reduce the variance of a randomized experiment? By taking the average of independent trials.

The following lemma states that the variance of the average of $n$ independent variables is $(1/n)$th their average variance. The key point is that averaging reduces variance. If we want to reduce the variance of an experiment by a factor of 10, simply repeat the experiment (independently) 10 times, and take the average!
**Lemma 7.5.** Let $X_1, \ldots, X_\ell$ be $\ell$ independent random variables. Let $\bar{X} = \frac{1}{\ell} \sum_{i=1}^\ell X_i$ be the average of the $X_i$'s. Let $\bar{V} = \frac{1}{\ell} \sum_{i=1}^\ell \text{Var}[X_i]$ be the average variance over the $X_i$'s. Then

$$\text{Var}[\bar{X}] \leq \frac{1}{\ell} \bar{V}.$$  

**Proof.** We have

$$\text{Var}[\bar{X}] = \frac{1}{\ell^2} \mathbb{E} \left[ (X_1 + \cdots + X_\ell - \mathbb{E}[X_1] - \cdots - \mathbb{E}[X_\ell])^2 \right]$$

$$\overset{(a)}{=} \frac{1}{\ell^2} \sum_{i=1}^\ell \sum_{j=1}^\ell \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])].$$

Here (a) applies linearity of expectation. Consider each term in the sum for fixed $i$ and $j$. If $i = j$, then

$$\mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])] = \mathbb{E}[(X_i - \mathbb{E}[X_i])^2] = \text{Var}[X_i]$$

If $i \neq j$, then

$$\mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])] \overset{(b)}{=} \mathbb{E}[X_i - \mathbb{E}[X_i]] \mathbb{E}[X_j - \mathbb{E}[X_j]] = 0,$$

where (b) is by independence of $X_i$ and $X_j$. Thus

$$\text{Var}[\bar{X}] = \frac{1}{\ell^2} \sum_{i=1}^\ell \text{Var}[X_i] = \frac{1}{\ell} \bar{V},$$

as desired. \(\square\)

**Variance reduction for distinct elements.** Let us return to the distinct elements problem. Recall that we have a random variable $X$ with expected value $1/(k+1)$, where $k$ is the number of distinct elements. $X$ also has variance $2k/(k+1)^2(k+2)$. Chebyshev’s inequality gives a bound of the form

$$\mathbb{P} \left[ |X - \mathbb{E}[X]| \leq \frac{\epsilon}{k+1} \right] \leq \frac{(k+1)^2}{\epsilon^2} \text{Var}[X] = \frac{k}{\epsilon^2(k+2)}, \quad (7.3)$$

which is isn’t very interesting since the RHS is greater than 1 for any interesting value of $\epsilon$. Now, however, we know how to reduce the variance.
Suppose we run $\ell$ independent copies of our experiment for $\ell \in \mathbb{N}$ TBD, producing $\ell$ independent random variables $X_1, \ldots, X_\ell$ each with expected value $1/(k+1)$ and variance $2k/(k+1)^2(k+2)$. Consider their average,

$$X \overset{\text{def}}{=} \frac{1}{\ell}(X_1 + \cdots + X_\ell).$$

Of course we have $E[X] = 1/(k+1)$. The real point is that the variance decreases at a linear rate:

$$\text{Var}[X] = \frac{1}{\ell} \cdot \frac{2k}{(k+1)^2(k+2)}.$$

In particular, when we repeat the analysis in (7.3) above, except now with respect to the average $X$, we get

$$\mathbb{P}\left[\left|\frac{1}{k+1} - \frac{1}{k+1} \cdot \epsilon \cdot \frac{1}{(k+2)}\right| \leq \frac{1}{\ell} \cdot \frac{k}{\epsilon^2(k+2)}\right].$$

We now have the additional parameter of $\ell$ to decrease the RHS. Setting $\ell \geq 1/\epsilon^2$, for example, we start to obtain interesting upper bounds that are less than 1. Suppose we want our probability of error to be $\delta$, where $\delta \in (0, 1)$. Setting $\ell = 1/\delta \epsilon^2$, we have

$$\frac{1 - \epsilon}{k+1} \leq X \leq \frac{1 + \epsilon}{k+1} \text{ with error probability } \leq \delta.$$

In turn, we have

$$(1 - 3\epsilon)k \leq \frac{1}{X} - 1 \leq (1 + 3\epsilon)k$$

with probability of error $\leq \delta$.

The new (idealized) algorithm is as follows. Suppose we want $(1 \pm \epsilon)$-multiplicative error with probability of error $\leq \delta$. We make $\ell = O(1/\epsilon^2 \delta)$ ideal hash functions $h_1, \ldots, h_\ell : [m] \rightarrow [0, 1]$. For each $i$, we let $X_i$ be the minimum hash value produced by $h_i$ over all the elements in the stream. We return

$$\frac{1}{X} - 1 \text{ where } X = \frac{1}{\ell} \sum_{i=1}^{\ell} X_i$$

is the average of the $X_i$'s seen so far.

Thus, ignoring the assumption that we have access to such hash functions, we need $O(1/\epsilon^2 \delta)$ space to obtain an $(1 \pm \epsilon)$-approximation to the number of distinct
elements with probability of error at most $\delta$. This is far better than where we started: in particular, we can get $(1 \pm \epsilon)$-error with constant probability in $O(1/\epsilon^2)$ space. Suppose, however, that we wanted extremely small probability of error. Can we get $(1 \pm \epsilon)$-error with probability at most $\delta$, in (say) $O(\ln(1/\delta)/\epsilon^2)$ space?

Put another way, right now we can decrease the probability of error linearly in the number of independent trials. Can we instead decrease the probability of error exponentially in the number of trials?

The median trick. In chapter 5, on random sums and graphs, we discuss the likelihood of various coin tossing experiments. For example, suppose we flipped $n$ independent coins, and consider the probability of getting at least 75% heads. Going from $n = 100$ coin tosses to $n = 1000$ coin tosses (a factor of 10) takes the odds of getting $\geq 75\%$ heads from $2^{-16.5}$ to $2^{-165}$ — much more than a 10-fold decrease. How can we use coin tosses to inspire a more efficient experiment design?

We currently know how to obtain an $(1 \pm \epsilon)$-approximation for the number of distinct elements with (any) constant probability in $O(1/\epsilon^2)$ space. Suppose we ran this algorithm 100 times independently (in parallel), where in each instance we take the average of enough trials so that the error probability is less than $.1$. We expect 90 of them to be accurate (enough), and since they are independent — as with independent coin tosses — the number of accurate experiments should be well concentrated. For example, more likely than not, at least 50 of the estimators have additive error $\leq \epsilon$.

Indeed, by the Chernoff bound (with $\epsilon = 4/9$), the probability of getting less than 50 correct instances is less than

$$e^{-(4/9)^2100/2} \leq .00005135.$$  

Thus, almost certainly, at least half of the estimators are correct. How can we pluck out one of the correct ones? Of the remaining ones, some are too big, and some are too small. We are looking for something in the “middle”.

The answer is not the mean. A single estimator that is atmospherically larger than all others will influence the mean too much. This is inherently unstable. Rather, we take the median of our estimators.

Indeed, consider the median estimate. The median estimate is too high only if at least half of the estimates are too high. The median estimate is too low only if at least half the estimates are too low. But, as our coin-flipping analysis has shown, the majority of our estimates is correct with high probability. In this event, the median is always correct.

---

4The answer is also not the “mode”, which is pretty useless in general.
More generally, we can repeat the algorithm \( O(\log(1/\delta)) \) times to achieve probability of error \( \leq \delta \).

This is called the \textit{median trick}, or the \textit{median of means}. By taking the \textit{median of means}, we can efficiently reduce the error probability at an \textit{exponential} rate. A colorful interface is given in fig. 7.1.

### 7.6 Distinct elements with pairwise independent hash functions

We have now developed a number of good ideas for the distinct elements problem. However we have continued to cheat in one critical way. Namely, we assumed access to ideal hash functions from \( h : [m] \to [0, 1] \).

Instead, suppose we used pairwise independent hash function \( h : [n] \to \{1/n^3, 2/n^3, \ldots, 1\} \). (That is, we create pairwise independent hash functions from \( [n] \) to \([n^3]\), and divide the output by \( n^3 \).) Rather than track the smallest hash, we will track the \( r \)th smallest hash for \( r = O(1/\epsilon^2) \). Let \( X \) be the \( r \)th smallest value be \( X \) for a single hash function. We return \( r/X \) as our estimate for the number of distinct elements.

For additional details, see [Nelson2020] or [ChekuriDistinctElements2014].

### 7.7 Takeaways

- Like the heavy hitters problem discussed previously, keeping track of the number of distinct elements is easy to do when the data fits in memory, but impossible to do exactly with sublinear space in streaming settings. Thus we consider approximations.

- Assuming access to an ideal and continuous hash function \( h \) into the interval \([0, 1]\), the minimum hash over all elements reflects the number of distinct elements.

- By taking the mean of a few independent trials, we can get small error with \textit{constant} probability of success. However, the mean is not concentrated well enough to be able to amplify it directly.

- Rather, we take the median of several independently sampled means to amplify the success probability at an exponential rate. In general, the median is more consistent than the mean.

- To argue that the median is concentrated, we applied the \textit{Chernoff inequality}, a generalization of the law of large numbers.
Median Trick

distribution $X$ w/ mean $\mu$

\[
\text{variance } \sigma^2
\]

$\lambda / l = O \left( \frac{\log (\frac{1}{\delta}) \sigma^2}{\epsilon^2 \mu^2} \right)$ i.i.d. instances

$X_1, \ldots, X_n \sim X$ one can compute (possibly biased) estimator $s$

\[
(1 - \varepsilon) \mu \leq Z \leq (1 + \varepsilon) \mu
\]

w/ prob $1 - \delta$.

$\frac{1}{\epsilon^2 \mu^2} \frac{\sigma^2}{\mu^2}$

Figure 7.1: The median trick.
7. Distinct elements

7.8 Additional notes and materials

Kent Quanrud

Fall 2022

• Ultimately, we produce an accurate and well-concentrated estimate of the number of distinct elements, but the estimate is not unbiased. This was also true of the count-min data structure for heavy hitters.

• To adapt the approach to pairwise independent hash functions, we instead find that the \( r \)th minimum hash, for \( r \approx 1/e^2 \), is more helpful.

7.8 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.

7.9 Exercises

Exercise 7.1. 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 \( \{1, \ldots, n\} \). Show that any algorithm that maintains the number of distinct elements exactly through the stream has to use at least \( n \) bits of memory.

**Hint:** argue that the algorithm must account for \( 2^n \) different possible states after the first \( n \) items in the stream.

Exercise 7.2. Let \( X \geq 0 \) be a continuous and nonnegative random variable. Prove that

\[
E[X] = \int_0^\infty P[X \geq t] dt.
\]

As a helpful hint, here are the first two steps towards deriving the claim.

\[
E[X] \overset{(a)}{=} \int_0^\infty tf(t) dt \overset{(b)}{=} \int_0^\infty \int_{s=0}^t f(t) ds dt.
\]

(a) is by definition of \( E[X] \). (b) expands out \( t \) to an integral.

Exercise 7.3. Prove lemma 7.1.

Exercise 7.4. Let \( X \) be as in lemma 7.1. Calculate (or estimate) \( E[(1/X) - 1] \), and show that it does not equal \( k \).
Exercise 7.5. Recall that in heavy hitters problem, the goal was to estimate the absolute frequency of each element (in $[n]$) up to an additive error of $\epsilon m$, 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 $\ell_2$-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 $x = 0^n$. The stream presents data of the form $(i, \Delta)$, 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 $\pm \|x\|_2$ 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 $h : [n] \rightarrow [w]$ be a pairwise independent hash function. Let $g : [n] \rightarrow \{-1, 1\}$ be a second pairwise independent hash function. The operations are as follows.

- For each update $(i, \Delta)$ 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, $E[y_i] = x_i$ for all $i$.)

2. 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 $\epsilon$, 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.)

3. Using the data structure designed above, design and analyze a data structure that, in $O(\log(n)/\epsilon^2)$, 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 $1/\text{poly}(n)$.)
Chapter 8

Dimensionality Reduction

8.1 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 [msccd].

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 $x$, $\|x\| = \sqrt{\sum x_i^2}$.

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 8.1 (johnson-lindenstrauss).** 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 $A \in \mathbb{N}^{k \times d}$ be a $k \times d$ randomized matrix where each coordinate is sampled as an independent Gaussian. Consider the randomly constructed linear map $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ defined by

$$f(x) = \frac{1}{\sqrt{k}} Ax.$$ 

Then with probability of error $\leq 1/\text{poly}(n)$, 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 8.1 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 8.1 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 8.3.

We remark that the embedding $A$ given in theorem 8.1 is not particularly compact, since it requires an independent Gaussian. This could be formidably expensive. Here one can instead replace the Gaussian entries with $\{-1, 0, 1\}$ entries generated by appropriate hash functions [achlioptas; dks; kane-nelson; cjn]. 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 [vershynin]). 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 [ailon-chazelle].

An important application of dimensionality reduction is in accelerating numerical algorithms on large matrices. See for example [woodruff; drineas-mahoney].

### 8.2 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 $\mu$ and $\sigma^2$. 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 parameters $\mu$ and $\sigma^2$ have simple interpretations.

**Fact 8.2.** Let $x \sim \mathcal{N}(\mu, \sigma^2)$. Then the mean and variance of $x$ are

$$\mathbb{E}[x] = \mu \text{ and } \text{Var}[x] = \sigma^2.$$  

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 8.3.** 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 8.4.** 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 8.5.** Let \( x \in \mathcal{N} \) and \( t < 1/2 \). Then

\[
\mathbb{E}[e^{tx^2}] = \frac{1}{\sqrt{1-2t}}.
\]

### 8.2.1 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 \( (\mu, \sigma^2) \)-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

\[
\mathbb{E}[\|x\|^2] = \sum_{i=1}^{k} \mathbb{E}[x_i^2] = \sum_{i=1}^{k} \text{Var}[x_i] = k \sigma^2.
\]

As a direct consequence of fact 8.5 above, the squared length \( \|x\|^2 \) of a Gaussian vector \( x \sim \mathcal{N}^k \) will be extremely well concentrated, as follows.

**Fact 8.6.** Let \( x \sim \mathcal{N}^k(0, \sigma^2) \) be a Gaussian vector. Let \( \alpha \geq 0 \).

1. If \( \alpha \leq 1 \), then

\[
\mathbb{P}[\|x\|^2 \leq \alpha \mathbb{E}[\|x\|^2]] \leq (\alpha e^{1-\alpha})^{k/2}.
\]
2. If \( \alpha \geq 1 \), then

\[
\mathbb{P}\left[\|x\|^2 \geq \alpha E[\|x\|^2]\right] \leq \left(\alpha e^{1-\alpha}\right)^{k/2}.
\]

**Proof.** Let us prove this fact because we only need the above facts to do so. Scaling \( x \) (and invoking fact 8.3), we can assume that \( x \in \mathcal{N}^k \) and \( \mu = E[\|x\|^2] = k \).

Let \( \alpha \in [0, 1] \). For \( t > 0 \), we have

\[
\mathbb{P}\left[\|x\|^2 \leq \alpha k\right] = \mathbb{P}\left[e^{-t\|x\|^2} \geq e^{-\alpha tk}\right] \leq \mathbb{E}\left[e^{-t\|x\|^2}\right] e^{\alpha tk} \leq \left(\prod_{i=1}^{k} \mathbb{E}\left[e^{-tx_i^2}\right]\right) e^{\alpha tk}
\]

\[
\leq (1 + 2t)^{k/2} \exp(\alpha tk) = \exp\left(k\left(\alpha t - \frac{1}{2} \ln(1 + 2t)\right)\right).
\]

(a) is by Markov’s inequality. (b) is by independence of the \( x_i \)'s (noting that \( \|x\|^2 = \sum x_i^2 \)). (c) is by fact 8.5. The (exponent of the) RHS is minimized by

\[
\alpha = \frac{1}{1 + 2t} \iff t = \frac{1 - \alpha}{2\alpha}.
\]

Plugging in \( t \) per (1) gives

\[
\mathbb{P}\left[\|x\|^2 \leq \alpha k\right] \leq \left(\alpha e^{1-\alpha}\right)^{k/2},
\]

as desired.

Now let \( \alpha \geq 1 \). For any \( t \in (0, 1/2) \), we have

\[
\mathbb{P}\left[\|x\|^2 \geq \alpha k\right] = \mathbb{P}\left[e^{t\|x\|^2} \geq e^{\alpha tk}\right] \leq \mathbb{E}\left[e^{t\|x\|^2} e^{-\alpha tk}\right]
\]

\[
\leq (1 - 2t)^{k/2} e^{-\alpha tk} = \exp\left(-\frac{k}{2}(2\alpha t + \ln(1 - 2t))\right)
\]

by (d) Markov’s inequality and (e) fact 8.5. 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

\[
\mathbb{P}\left[\|x\|^2 \geq \alpha k\right] \leq \left(\alpha e^{1-\alpha}\right)^{k/2},
\]

as desired. \( \square \)
8. Dimensionality Reduction

8.3. Random Projections

An important case is where $\alpha = (1 \pm \epsilon)$ and $\epsilon > 0$ is close to 0. Then fact 8.6 implies the following.

**Lemma 8.7.** Let $x \sim \mathcal{N}^{k}(0, \sigma^2)$ be a Gaussian vector. Let $\epsilon \in (0, 1]$. Then

(i) $\mathbb{P}[\|x\|^2 \leq (1 - \epsilon) \mathbb{E}[\|x\|^2]] \leq e^{-c\epsilon^2k/2}$ for $c = 1 - \ln(2) \approx .307$.

(ii) $\mathbb{P}[\|x\|^2 \geq (1 + \epsilon) \mathbb{E}[\|x\|^2]] \leq e^{-\epsilon^2k/4}$.

**Proof.** By scaling, we can assume that $\sigma^2 = 1$ and $\|x\|^2 = 1$. We have

\[ \mathbb{P}[\|x\|^2 \geq (1 + \epsilon)k] \leq (1 + \epsilon)e^{-\epsilon^2k/4} \leq e^{-\epsilon^2k/4}. \]

Here (a) is by fact 8.6. (b) is because $1 + \epsilon \leq e^{-\epsilon^2k/4}$ for $\epsilon \in [0, 1]$.

Likewise, we have

\[ \mathbb{P}[\|x\|^2 \leq (1 - \epsilon)k] \leq (1 - \epsilon)e^{-\epsilon^2k/4}. \]

(c) is by fact 8.6. (d) is because $1 - \epsilon \leq e^{-\epsilon^2k/4}$ for $\epsilon \in [0, 1]$.

8.3 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 8.8.** 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 $x$,

\[ \mathbb{P}[ (1 - \epsilon)\|x\|^2 \leq \frac{1}{k} \|Ax\|^2 \leq (1 + \epsilon)\|x\|^2] \geq 1 - 2e^{-k/8}. \]

**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 $i$th 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 8.3 and 8.4, $\langle a_i, x \rangle$ is a Gaussian random variable with mean 0 and variance

\[ \sum_{j=1}^{n} \text{Var}[x_j A_{ij}] = \sum_{j=1}^{n} x_j^2 = 1. \]

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 8.7. \qed
Consider theorem 8.1 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 8.8, for each \( x \in P \), we have

\[
(1 - \epsilon)\|x\|^2 \leq \|f(x)\|^2 \leq (1 + \epsilon)\|x\|^2
\]

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 8.1 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.

### 8.4 Gaussians

Based on theorem 8.1, a distribution satisfying facts 8.2–8.5 (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

\[
f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x - \mu}{\sigma})^2}.
\]

(By lemma 8.9 below, this random variable indeed has mean \( \mu \) and variance \( \sigma^2 \).) We let \( \mathcal{N}(\mu, \sigma^2) \) to denote the normal distribution with mean \( \mu \) and variance \( \sigma^2 \), 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.
8. Dimensionality Reduction

8.4. Gaussians

8.4.1 Some preliminary calculus

Lemma 8.9. Let $\mu \in \mathbb{R}$, $\sigma > 0$, and

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Then we have the following.

1. $\int_{-\infty}^{\infty} f(x) = 1$.
2. $\int_{-\infty}^{\infty} xf(x) = \mu$.
3. $\int_{-\infty}^{\infty} (x-\mu)^2 f(x) = \sigma^2$.

Proof. We consider the normalized case $\mu = 0$ and $\sigma = 1$. The general case follows by appropriate change of variables. We have

$$\left(\int_{-\infty}^{\infty} e^{-x^2/2} dx\right)^2 = \int_{-\infty}^{\infty} e^{-(x^2+y^2)/2} dxdy = \int_0^{2\pi} \int_{-\infty}^{\infty} re^{-r^2/2} drd\theta = 2\pi \int_{0}^{\infty} re^{-r^2/2} dr = 2\pi.$$

Taking the square root of both sides gives the first claim. For the second claim, we have

$$\int_{-\infty}^{\infty} xe^{-x^2/2} dx = [e^{-x^2/2}]_{-\infty}^{+\infty} = 0.$$

For the third claim, we have

$$\int_{-\infty}^{\infty} x e^{-x^2/2} dx \overset{(a)}{=} [-xe^{-x^2/2}]_{-\infty}^{+\infty} + \int_{-\infty}^{\infty} e^{-x^2/2} = 1$$

by (a) integration by parts.

Lemma 8.9 immediately implies both fact 8.2 and fact 8.3, which we restate for convenience.

Fact 8.2. Let $x \sim \mathcal{N}(\mu, \sigma^2)$. Then the mean and variance of $x$ are

$$\mathbb{E}[x] = \mu \text{ and } \text{Var}[x] = \sigma^2.$$

Fact 8.3. 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 $x + \alpha \sim \mathcal{N}(\alpha, \sigma^2)$. 

136
8.4.2 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 8.10.** $f(x) = (2\pi)^{-n/2}e^{-\langle x,x \rangle/2}$.

**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. \qed

**Lemma 8.11.** 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

$$f(Ux) \overset{(a)}{=} (2\pi)^{-n/2}e^{-\langle Ux,Ux \rangle/2} \overset{(b)}{=} (2\pi)^{-n/2}e^{-\langle x,x \rangle/2} \overset{(c)}{=} f(x),$$

where (a) is by lemma 8.10, (b) is because $U^TU = I$, and (c) is by lemma 8.10. \qed

**Lemma 8.12.** Let $x \sim \mathcal{N}^n$ and $u \in \mathbb{R}^n$. Then $\langle u, x \rangle \sim \mathcal{N}(0, \|u\|^2)$.

**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

$$\langle u, x \rangle = \langle U^T e_1, x \rangle = \langle e_1, Ux \rangle = \langle e_1, x \rangle \overset{(a)}{\sim} \mathcal{N},$$

where (a) is by lemma 8.11. \qed

Fact 8.4, which says that Gaussians sum nicely, now follows by combination of fact 8.3 and lemma 8.12. We restate fact 8.4 for convenience and leave the proof to the reader.

**Fact 8.4.** If $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, then $x_1 + x_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. 137
8.4.3 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 8.1.

**Fact 8.5.** Let $x \in \mathcal{N}$ and $t < 1/2$. Then

$$
E[e^{tx^2}] = \frac{1}{\sqrt{1 - 2t}}.
$$

**Proof.** We have

$$
E[e^{tx^2}] = \int_s e^{ts^2} P[x = s] \overset{(a)}{=} \frac{1}{\sqrt{2\pi}} \int_s e^{(2t-1)s^2/2} = \frac{1}{\sqrt{2\pi}} \int_s e^{-s^2/2\sigma^2} \quad \text{for } \sigma = 1/\sqrt{1 - 2t} \overset{(b)}{=} \frac{1}{\sqrt{1 - 2t}}.
$$

Here (a) plugs in the density function from equation (8.3). (b) is by lemma 8.9.1 w/r/t the density function for $\mathcal{N}(0, \sigma^2)$.

8.5 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.

8.6 Exercises

**Exercise 8.1.** Using only fact 8.3, show that for $x \sim \mathcal{N}(\mu, \sigma^2)$ and $\alpha \in \mathbb{R}$,

$$
\alpha x \sim \mathcal{N}(\alpha \mu, \alpha^2 \sigma^2).
$$
Exercise 8.2. 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 8.3. Let $P \subseteq \mathbb{R}^d$ be a set of $n$ points. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ be a random projection with $k = O(\log(n)/\epsilon^2)$ (per theorem 8.1). 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

$$|\langle f(x), f(y) \rangle - \langle x, y \rangle| \leq \frac{\epsilon}{2}(\|x\|^2 + \|y\|^2)$$

for all $x, y \in P$.

*Hint: What is $\|x + y\|^2 - \|x - y\|^2$?*
Chapter 9

Locality sensitive hashing and approximate nearest neighbors

9.1 Nearest neighbor search

The previous discussion on dimensionality reduction introduced many natural settings where data can be represented as numerical vectors in an high-dimensional space. Surprising, by simply projecting onto randomized Gaussian vectors, one can embed an \( n \)-point data set into \( O(\log(n)/\epsilon^2) \) dimensions while preserving all pairwise distances up to an \((1 \pm \epsilon)\)-multiplicative factor.

A natural query in many domains is similarity search where we want to preprocess a large collection of items such that, given a query in the form of an 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.

A simple example might be where we have a collection of \( n \) real numbers. Given another number as a query, we want to retrieve the closest number in our collection to the query number. For this problem, we would sort our collection of numbers and store them in an array. Given a query \( q \in \mathbb{R} \), we run binary search on our array to find the first number smaller and bigger than \( q \). We return the closer of the two. This approach would take \( O(\log(n)) \) 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 \( d \). If the data lives in very few dimensions - like 3 dimensions, as in many physical situations - this is still very good.

Continuing the previous discussion on dimensionality reduction, here we are interested in nearest neighbor search with high dimensional data. We have a collection of \( n \) points \( P \subset \mathbb{R}^d \) that we can preprocess to build some kind of data structure. Each query is in the form of another point \( z \in \mathbb{R}^d \), and the goal is to output the point \( x \in P \) closest to \( z \) by some metric. Here we will consider two settings.

1. The Euclidean distance, \( \|x - z\| \).

2. The angle between \( x \) and \( z \) (as a real value between 0 and \( \pi \)).
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 9.2 and Euclidean nearest neighbors in section 9.3.

A natural idea, at least for Euclidean distance, is to use the dimensionality reduction techniques from the previous discussion to reduce to $O(\log(n))$ dimensions, and apply the low-dimensional data structures. But the exponential dependence in the dimension means that even $O(\log(n))$ dimensions – which is fairly small by our standards – will still require $n^{O(1)}$ time and space.

**From nearest neighbor to close enough.** Fix $\sigma > 1$. Enumerating distances to the nearest neighbor by powers of $\sigma$ allows us to reduce to the following problem with only logarithmic overhead:

Let a target distance $r > 0$ be fixed. Given a query point $z$, either output a point at distance $\leq \sigma r$, or declare that all points have distance $\geq r$.

It also suffices, up to logarithmic factors, to succeed with constant probability. We can amplify to high probability with repetition.

**When hash collisions are good.** The algorithms we develop are based on hashing, although the intuition is the opposite of previous discussions on hashing. Previously, in applications such as heavy hitters and hash tables, hashing was used to randomly spread out the elements. Here, we will design hash functions where closer points are more likely to collide. This technique is called locality sensitive hashing [indyk-motwani].

The high level strategy is to build a hash table over all 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. In the algorithms we discuss, this algorithm will 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.

### 9.2 Locality Sensitive Hashing for Angles

We consider the setting of angular distance, which is commonly used when the point set $P$ lies on the hypersphere $S^{d-1} = \{x \in \mathbb{R}^d : \|x\| = 1\}$. In this case a common measure of distance 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 $\pi$. We may assume that

\[^1\text{Here the author would like to propose the term clashing.}\]
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 [charikar].

Let \( \theta \in [0, \pi] \) be a fixed angle. In this discussion, given a query point \( y \), our goal is to either output a point \( x \in P \) with \( \angle(x, y) \leq 4\theta \), or 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 \( h : \mathbb{S}^{d-1} \rightarrow \{-1, 1\}^k \) 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 \( h : \mathbb{S}^{d-1} \rightarrow \{-1, 1\}^k \) by
\[
    h_i(x) = \text{sign}(\langle g_i, x \rangle)
\]
for each coordinate \( i \).

We will leverage the following fact that is entirely based on the rotational invariance of \( \mathcal{N}^d \).

**Lemma 9.1.** Let \( x, y \in P \), and let \( g \sim \mathcal{N} \). Then
\[
P[\text{sign}(\langle g, x \rangle) \neq \text{sign}(\langle g, y \rangle)] = \frac{\angle(x, y)}{\pi}.
\]

**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.

We are particularly interested in the angle of the random vector \( g \). Since \( \mathcal{N}^2 \) is rotationally symmetric, it is equally likely to take any particular angle with equal probability. Out of a total of \( 2\pi \), there are two regions of angle \( \angle(x, y) \) in which \( \text{sign}(\langle g, x \rangle) \neq \text{sign}(\langle g, y \rangle) \), here marked with a red X.

Now, let \( k = \frac{\pi \log(n)}{2\theta} \). If \( \angle(x, y) < \theta \), then
\[
P[h(x) = h(y)] = \left(1 - \frac{\angle(x, y)}{\pi}\right)^k \approx e^{-\frac{\angle(x, y)}{\pi}k} = e^{-\log(n)/2} = \frac{1}{\sqrt{n}}.
\]

On the other hand, if \( \angle(x, y) > 2\theta \), then
\[
P[h(x) = h(y)] = \left(1 - \frac{\angle(x, y)}{\pi}\right)^k \leq e^{-\angle(x, y)k/\pi} \leq e^{-\log(n)} = \frac{1}{n}.
\]

Thus, when querying a point \( y \):
1. If there is a point \( x \in P \) with \( \angle(x, y) \leq \theta \), then it will collide with probability (approximately) \( \geq 1/\sqrt{n} \).

2. We expect to collide with at most 1 point \( x \in P \) such that \( \angle(x, y) \geq 2 \theta \).

If we repeat the experiment \( O(\sqrt{n} \log(n)) \) times, then with high probability we will find a close neighbor if one exists. The query time is \( O(d\sqrt{n} \log(n)) \) in expectation, because each time we expect 1 “junk” neighbor on average.

**Theorem 9.2.** With \( O\left(dn^{3/2} \text{ polylog}(n)\right) \) preprocessing time and space, one can query for 2-approximate nearest neighbors w/r/t angular distance in \( O(d\sqrt{n} \text{ polylog}(n)) \) expected randomized time and with high probability.

### 9.3 Locality Sensitive Hashing for Euclidean Distance

We now consider \( \sigma \)-approximate nearest neighbors in Euclidean metrics, for fixed \( \sigma > 1 \). As mentioned above, it suffices to consider the simpler setting where there is a fixed target distance \( r \). We want to preprocess a set of \( n \) points \( P \subseteq \mathbb{R}^d \) as to quickly answer the following query with constant probability of success:

Given a query point \( z \in \mathbb{R}^d \), either find a point \( x \in P \) with \( \|x - z\| \leq \sigma r \), or declare that there are no points \( x \in P \) with \( \|x - z\| \leq r \).

Our goal is to develop a locality-sensitive hash function\(^2\) \( h : \mathbb{R}^d \rightarrow \mathbb{Z}^k \) for Euclidean distance. We will design \( h \) such that the collision probabilities of two points \( x, y \in \mathbb{R}^d \) depend only on the ratio \( \|x - y\| \) to \( r \). Therefore, we can also rescale and assume that \( r = 1 \).

Given a query point \( z \in \mathbb{R}^d \), either find a point \( x \in P \) with \( \|x - z\| \leq \sigma \), or declare that there are no points \( x \in P \) with \( \|x - z\| \leq 1 \).

### 9.3.1 Random line embeddings and buckets

In this section, we first define a hash function \( h : \mathbb{R}^d \rightarrow \mathbb{Z} \) that outputs a single hash code. Later we will consider a \( k \)-coordinate hash where each coordinate is an independent copy of the single coordinate hash we consider here.

We define \( h : \mathbb{R}^d \rightarrow \mathbb{Z} \) by the function

\[
h(x) = \lfloor \langle g, x \rangle + \alpha \rfloor,
\]

\(^2\)a clash function?
where $g \sim \mathcal{N}^d$ and $\alpha \in [0, 1]$ uniformly at random. Geometrically, we can interpret $h(x)$ as randomly projecting $x$ onto a line, and then bucketing the points in intervals of length 1. The random value $\alpha \in [0, 1]$ translates the buckets randomly.

**Lemma 9.3.** Let $x, y \in \mathbb{R}^d$.

$$
\mathbb{P}[h(x) = h(y) \mid g] = \max\{0, 1 - |\langle x - y, g \rangle|\}.
$$

**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 $\alpha$) falls between $\langle g, x \rangle$ and $\langle g, y \rangle$. If $|\langle g, x \rangle - \langle g, y \rangle| \geq 1$, this always happens. Otherwise, it happens with probability $|\langle g, x \rangle - \langle g, y \rangle|$.

**Lemma 9.4.** Let $x, y \in \mathbb{R}^d$, and let $f(t)$ be the density function of the standard Gaussian $\mathcal{N}$.

$$
\mathbb{P}[h(x) = h(y)] = 2 \int_0^1 f(||x - y||t)(1 - t)
$$

**Proof.** Recall that $|\langle x - y, g \rangle| \sim \mathcal{N}(0, \sigma^2)$. In particular, $\langle x - y, g \rangle$ has density function $f(\sigma t)$, and $|\langle x - y, g \rangle|$ has density function $2f(\sigma t)$. We have

$$
\mathbb{P}[h(x) = h(y)] = 2 \int_0^1 \mathbb{P}[h(x) = h(y) \mid |\langle g, x - y \rangle| = t]f(||x - y||t) \, dt
$$

$$
= 2 \int_0^1 (1 - t)f(||x - y||t) \, dt,
$$

as desired. \qed

**Remark 9.5.** The collision probability obtained in lemma 9.4 is a function of $\sigma$, which is the ratio between $||x - y||$ and the target distance (here normalized to 1). This justifies our normalization. Note also that $\mathbb{P}[h(x) = h(y)]$ is decreasing in $||x - y||$.

lemma 9.4 derives 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

$$
p = 2 \int_0^1 (1 - t)f(t) \, dt
$$

144
be a lower bound on the collision probability when \( \|x - y\| \leq 1 \). Let

\[ q = 2 \int_0^1 (1 - t) f(\sigma t) \, dt \]

be an upper bound on the collision probability when \( \|x - y\| \geq \sigma \). We note that \( p \) is a fixed constant, about .368746....

Since \( f(\sigma t) \) is decreasing in \( \sigma \), \( p > q \). That is, close points are more likely to collide than far points. To what extent? It turns out that the gap is not enough to simply use \( h \) as a locality sensitive hash function. But we can remedy this simply by amplification, as follows.

### 9.3.2 Amplifying the gap

Let \( k \in \mathbb{N} \) be a parameter TBD. We define a hash function

\[ h : \mathbb{R}^d \rightarrow \mathbb{R}^k \]

by defining each coordinate \( h_i(x) \) according to the single-coordinate hash function in section 9.3.1; 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 9.4, we have

\[ \mathbb{P}[h(x) = h(y)] = \left( 2 \int_0^1 (1 - t) f(\|x - y\||t) \, dt \right)^k. \]

In particular, recalling the values of \( p \) and \( q \) as above, we have

\[ \mathbb{P}[h(x) = h(y)] \geq p^k \text{ when } \|x - y\| \leq 1 \]

and

\[ \mathbb{P}[h(x) = h(y)] \leq q^k \text{ when } \|x - y\| \geq \sigma. \]

Now, \( k \) decreases both \( p^k \) and \( q^k \) decreases 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.\(^3\) On the other hand, as \(q^k\) decreases, the number of hash collisions with bad points also reduces. Ultimately, the algorithm pays a running time proportional to the total number of bad collisions as it scans the lists in the hash bucket as it searches for a good quantity. Overall, the ratio \((q/p)^k\) decreases, so to at least some extent \(k\) is useful.

Ultimately, the quantity we want to minimize is

\[
k \left( \frac{1}{p} \right)^k + \left( \frac{q}{p} \right)^k n.
\]

The \((1/p)^k\) term represents having the compute \(\ell = (1/p)^k\) hash codes. The \(\left( \frac{q}{p} \right)^k n\) represents the expected number of hash collisions across all \(\ell\) instances with bad elements. The above quantity models the running time up to an additional factor of \(d\), incurred from either hashing the query point or comparing the distance between the query point and a point in the same hash bucket. We can rewrite the above as

\[
\left( \frac{1}{p} \right)^k \left( k + q^k n \right).
\]

Choose \(k = \log(1/n)/\log(q) = \log(n)/\log(1/q)\). 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. In fact, one can show that it is about \(1/1 + \epsilon\).

**Theorem 9.6.** One can compute an \((1 + \epsilon)\)-approximate nearest neighbor w/r/t Euclidean distance with high probability in \(\tilde{O}(n^{\rho(\sigma)})\) randomized time, where

\[
\rho(\sigma) = \frac{\log(1/p)}{\log(1/q)}, \quad p = 2 \int_0^1 (1 - t) f(t) dt \approx 0.368746, \quad q = 2 \int_0^1 (1 - t) f(\sigma t) dt,
\]

and \(f(t) = e^{-t^2/2}/\sqrt{2\pi}\) is the density function of the standard Gaussian.

Below, we show that for \(\sigma^2 \leq 1.6\), we have

\[
\rho(\sigma) \leq 1/\sigma^2.
\]

\(^3\)Indeed, the probability of failing to find the good neighbor in each of \(\ell\) constructions, each of which has \(k\) hash coordinates, is

\[
(1 - p^k)^\ell \approx e^{-p^k \ell}.
\]
9. Locality sensitive hashing and approximate nearest neighbors

9.3. Locality Sensitive Hashing for Euclidean Distance

9.3.3 Analyzing $\rho(\sigma)$

For any fixed value of $\sigma > 1$, one can run a numerical computation to get an accurate estimate of $\rho(\sigma)$. See for example [diim]. Below, we analyze $\rho(\sigma)$ and obtain some upper bounds on $\rho$. While these bounds are of course not as tight as a computer simulation, they reveal some intuition for how the value of $\rho(\sigma)$ varies with $\sigma$.

**Lemma 9.7.** Suppose $\sigma^2 = 1 + \epsilon$ for $\epsilon \in (0, 1)$. Then

$$q \geq (1 - \epsilon)^2 p.$$ 

In particular,

$$\rho(\sigma) \geq \frac{\log(1/p)}{\log(1/p) - 2 \log(1 - \epsilon)}.$$ 

**Proof.** We have

$$\sqrt{2\pi}(p - q) = \int_0^1 (1 - t)\left(e^{-t^2/2} - e^{-\sigma^2 t^2/2}\right) dt$$

Say $\sigma^2 = (1 + \epsilon)$ with $\epsilon$ sufficiently small. Then for all $t \in [0, 1]$, we have

$$e^{-t^2/2} - e^{-(1+\epsilon)t^2/2} = e^{-t^2/2}(1 - e^{-\epsilon t^2/2}) = e^{-t^2/2}\left(\epsilon t^2/2 - \left(\epsilon t^2/2\right)^2/2\right) \leq \epsilon(1 - \epsilon)e^{-t^2/2}.$$ 

Thus

$$\int_0^1 (1 - t)\left(e^{-t^2/2} - e^{-2\sigma^2 t^2/2}\right) dt \leq \epsilon(1 - \epsilon) \int_0^1 (1 - t)e^{-t^2/2} dt = \epsilon(1 - \epsilon)\sqrt{2\pi}p.$$ 

Thus

$$\sqrt{2\pi}(p - q) \leq \epsilon(1 - \epsilon)\sqrt{2\pi}p;$$

in turn;

$$(1 - \epsilon)^2 p \leq q.$$ 

**Lemma 9.8.** Let $\sigma^2 \leq 1.6$. Then

$$\rho(\sigma) \leq \frac{1}{\sigma^2}.$$
9. Locality sensitive hashing and approximate nearest neighbors

9.4. Exercises

Kent Quanrud
Fall 2022

Proof. We continue from the conclusion of lemma 9.7 with \( \epsilon = \sigma^2 - 1 \). One can directly compute that \( \log(1/p) \) is a constant, about (and slightly greater than) .77. On the other hand, \( \log(1 - \epsilon) \approx -\epsilon \) for small \( \epsilon \); in particular, \(-2\log(1 - \epsilon) \geq -\log(1/p)\epsilon \) for \( \epsilon \leq .6 \). Thus

\[
\rho(\sigma) \leq \frac{\log(1/p)}{\log(1/p) - 2\log(1 - \epsilon)} \leq \frac{\log(1/p)}{(1 + \epsilon)\log(1/p)} = \frac{1}{1 + \epsilon} = \frac{1}{\sigma^2}.
\]

9.4 Exercises

Exercise 9.1. In this exercise, we will develop a 2-approximation LSH scheme for bit strings \( s \in \{0, 1\}^d \) of a fixed length \( d \) w/r/t 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 \( P[h(s) = h(t)] \), 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 w/r/t Hamming distance.
Chapter 10

Sampling geometric range spaces

10.1 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.) 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 \(P\) contain?

3. Does \(r\) contain at least an \(\epsilon\)-fraction of \(P\), for given \(\epsilon\)?

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:

\[
\mu(r) = \frac{|r \cap P|}{|r|}.
\]

One can interpret \(\mu(r)\) as the probability that a random point from \(P\) lies 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 \(\mu_Q\) denote the measure with respect to \(Q\):

\[
\mu_Q(r) = \frac{|r \cap Q|}{|Q|}.
\]

We say that \(Q\) is an \(\epsilon\)-sample for \((P, R)\) if it approximates the measure up to an \(\epsilon\)-additive error; that is,

\[
|\mu(r) - \mu_Q(r)| \leq \epsilon \text{ for all } r \in R.
\]

149
Q is an \(\epsilon\)-net if it hits all ranges with measure at least \(r\):

\[ \mu_Q(r) > 0 \text{ for all } r \in R \text{ with } \mu(r) > \epsilon. \]

Note that an \(\epsilon\)-sample is stronger than an \(\epsilon\)-net. \(\epsilon\)-nets are useful in situations where you want to identify “heavy-hitter” range spaces.

Computing a small \(\epsilon\)-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 \(Q \subseteq P\), \(Q\) cannot be an \(\epsilon\)-sample because in particular it will fail for the range \(r = P \setminus Q\). Similarly one can argue that any \(\epsilon\)-net \(Q\) of \(P\) must have at least \((1 - \epsilon)|P|\) points, which is not much smaller.

Still there are many natural range spaces, especially in computational geometry, that are not nearly as complex as \(2^P\). Above we mentioned the setting of disks in the plane, which arises in many geometric applications. The intersection of a disk with the point set \(P\) gives a subset of \(P\), but in general, disks cannot induce all possible subsets. 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 in our sampling.

For a fixed range space \((P, R)\), the growth function \(g : \mathbb{N} \to \mathbb{N}\) is defined by For \(k \in \mathbb{N}\), \(g(k)\) is defined as the maximum, over all \(Q \subseteq P\), of the number of distinct subsets induced by intersecting \(Q\) with the ranges in \(R\):

\[ g(k) = \max_{Q \subseteq P} |\{r \cap Q : r \in R\}|. \]

\((P, R)\) 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. 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 and the shattering-dimension, which we explain in greater detail later.

In the following theorems, let \((P, R)\) be a range space with polynomial growth \(d\), and \(|P|\) finite. Let \(\epsilon, \delta \in (0,1)\), and let \(Q \subseteq P\) be a random sample of \(P\). We want to understand how big \(Q\) should be so that it is either an \(\epsilon\)-sample or an \(\epsilon\)-net with some probability. For \(\epsilon\)-samples we have the following [VC71].

**Theorem 10.1.** If \(|Q| \geq \frac{C}{\delta^2} (d \log(d/\epsilon) + \log(1/\delta))\), then \(Q\) is an \(\epsilon\)-sample with probability at least \(1 - \delta\). Here \(C\) is a universal constant.
10. Sampling geometric range spaces

10.2. Proof of the $\epsilon$-sample theorem

Note that $|Q|$ is independent of $|P|$. For $\epsilon$-nets an even smaller sample suffices [HW87]:

**Theorem 10.2.** If $Q$ is a random sample with repetition with $|Q| \geq C(\ell \log(1/\epsilon) + \log(1/\delta))$, then $Q$ is an $\epsilon$-net with probability at least $1 - \delta$. Here $C$ is a universal constant.

10.2 Proof of the $\epsilon$-sample theorem

In the section we prove the $\epsilon$-sample theorem. The following lemma states the theorem in a slightly more general form. The $\epsilon$-sample theorem follows from setting $\ell = O(\ell \log(d/\epsilon) + \log(1/\delta)/\epsilon^2)$ and applying the multiplicative Chernoff bound.

**Lemma 10.3.** Let $(P, R)$ be a range space with growth function $g(n)$. Let $\epsilon, \delta \in (0, 1)$.

Let $\ell$ be sufficiently large with the following property. For any set $Q_0 \subset P$ inducing measure $\mu_0$, and any random sample $Q_1$ from $Q_1$ of $\ell$ points inducing measure $\mu_1$, and all $r \in R$, we have

$$\Pr[|\mu_0(r) - \mu_1(r)| \geq \epsilon/4] \leq \frac{\delta}{4g(2\ell)}.$$ 

Then a random sample $Q_1$ of size $\ell$ is an $\epsilon$-sample with probability $1 - \delta$.

**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 $\Pr[A] < \delta$.

Let $Q_2$ be a second, independent random sample of the same size. Let $B$ be the event that $\mu_1$ and $\mu_2$ disagree on $r$ by more than $\epsilon/2$:

$$B \overset{\text{def}}{=} \text{the event that } |\mu_1(r) - \mu_2(r)| > \epsilon/2 \text{ for some } r \in R.$$ 

We claim that $\Pr[A] \leq 2\Pr[B]$. To this end, we first write

$$\Pr[B] \geq \Pr[A, B] \geq \Pr[B \mid A] \Pr[A]. \quad (10.1)$$ 

Now, in event $A$, a range $r$ is incorrectly measured by $Q_1$ by more than $\epsilon$. That particular range $r$ is correctly measured by $Q_2$ to within $\epsilon/2$ with probability at least $1/2$. In this case we have

$$|\mu_1(r) - \mu_2(r)| \geq |\mu(r) - \mu_1(r)| - |\mu(r) - \mu_2(r)| > \epsilon - \epsilon/2 = \epsilon/2,$$

151
hence event $B$. Thus $P[B \mid A] \geq 1/2$, which gives $P[B] \geq P[A]/2$ when plugged into (10.1).

Thus an upper bound on $P[B]$ gives an upper bound on $P[A]$ up to a factor of 2. To upper bound $P[B]$, suppose we generate $Q_1$ and $Q_2$ alternatively as follows.

1. Sample $2\ell$ points $Q_0$ for $P$.
2. Randomly partition $Q_0$ in half. Let $Q_1$ be one half and let $Q_2$ be the other.

Since

$$P[B] = \sum_{Q_0} P[Q_0] P[B \mid Q_0] \leq \max_{Q_0} P[B \mid Q_0],$$

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 $\mu_0$ be the measure with respect to $Q_0$. For each $r$, we have

$$P[|\mu_1(r) - \mu_0(r)| \geq \epsilon/4] \leq \frac{\delta}{4m}$$

as well as

$$P[|\mu_2(r) - \mu_0(r)| \geq \epsilon/4] \leq \frac{\delta}{4m}.$$

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 $P[B] \leq \delta/2$, completing the proof. $\square$

10.3 Additional notes and materials

For additional background on learning theory, see [Schapire2019].
Chapter 11

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 looking for 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

$$\frac{1}{n} \cdot \frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}} \leq \frac{c(\delta(S))}{|S||\bar{S}|} \leq 2 \cdot \frac{c(\delta(S))}{2 \min\{|S|, |\bar{S}|\}}$$

because $n/2 < \max\{|S|, |\bar{S}|\} \leq n$. That is, up to a constant factor, we are trying to minimize the ratio

$$\frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}}.$$  

153
Here we clearly see the expense of choosing a very small set \( S \).

In this chapter, we describe a very influential result of **leighton-rao** that obtains a \( O(\log(n)) \) approximation ratio for the uniform sparsest cut problem. Their algorithm is based on applying region growing to the metric induced by the dual LP. We then discuss a 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.

### 11.1 The LP

**leighton-rao**’s algorithm \([\text{leighton-rao}]\) is based on rounding an LP relaxation of the sparsest cut problem with region growing. 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:

\[
\begin{align*}
\text{minimize} & \quad \sum_{e=\{u,v\} \in E} c(e)d(u,v) \\
\text{over all metrics} & \quad d : V \times V \to \mathbb{R}_{\geq 0} \\
\text{s.t.} & \quad \sum_{\{s,t\}} b(s,t)d(s,t) \geq 1. \tag{11.1}
\end{align*}
\]

### 11.1.1 The dual

To obtain the dual, it is helpful to rewrite (11.1) as a pure covering problem. Recall the correspondence between metrics and edge lengths, via shortest path distances. Then (11.1) 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 $\mathcal{P}_* \overset{\text{def}}{=} \prod_{(s,t) \in \mathcal{P}_{s,t}}$ denote the family of all path bundles. Then we can express the problem above as follows.

$$\minimize \sum_{e \in E} c(e)y(e) \quad \text{over } y : E \to \mathbb{R}_{\geq 0}$$

$$\text{s.t. } \sum_{\{s,t\}} b(s, t) \sum_{e \in P_{s,t}} y(e) \geq 1 \text{ for all } P \in \mathcal{P}_*. \quad (11.2)$$

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.

(11.2) covers path bundles with edges; thus, the dual packing LP packs path bundles into edges.

$$\maximize \sum_{P \in \mathcal{P}_*} x(P) \quad \text{over } x : \mathcal{P}_* \to \mathbb{R}_{\geq 0}$$

$$\text{s.t. } \sum_{P \in \mathcal{P}_*} x(P) \sum_{\{s,t\} : e \in P_{s,t}} b(s, t) \leq c(e) \text{ for all } e \in E. \quad (11.3)$$

In (11.3), each path bundle $P$ represents a choice of paths for every $(s, t)$-pair to concurrently route the demands $b$. So (11.3) 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.

### 11.2 Rounding via region growing (for uniform demands)

leighton-rao addressed the setting of uniform demands $b = 1$, as well as “product demands” where the demands are of the form $b(u, v) = \pi(u)\pi(v)$ for a fixed vector $\pi \in \mathbb{R}^V_\geq 0$. In this section we assume uniform demands, $b = 1$. Then the sparsity of a set $S$ has the cleaner form of

$$\frac{c(\delta(S))}{|S||S|}.$$ 

Let $d$ be an optimum metric to the metric relaxation (11.1). For uniform demands we may assume that $d(u, v) \leq 1$ for all $u, v$. The goal is to compute a set of vertices $S$
Our analysis is divided into two cases. The first, dubbed the “concentrated” case, is when there is a high concentration of points within a small ball of radius \( r_0 \approx 1/n^2 \). The second “non-concentrated” case is when this does not occur. In the concentrated case we show that the line embedding from the center of the concentrated ball induces a sparse cut. In the non-concentrated case we partition the graph into pieces with a region-growing technique; the non-concentrated setting implies that these pieces are individually small enough to be reassembled to give a sparse cut.

For the remainder of this section, let \( r_0 = 1/2n^2 \).

### 11.2.1 Concentrated case.

First we consider the concentrated case.

**Lemma 11.1.** Suppose there exists a vertex \( s \) such that \( |B(s, r_0)| \geq 2n/3 \). Then there exists \( r \geq r_0 \) such that \( B(s, r) \) has sparsity \( O(1) \OPT \).

**Proof.** Suppose not. Then for all \( q > 0 \), we have

\[
\frac{c(\delta(B(s, r_0 + q)))}{|S||S|} \leq O(\log(n)) \OPT. \tag{11.1}
\]

for a fixed universal constant \( c > 0 \) (to be determined). Here (a) observes that \( |B(s, r_0 + q)| \geq |B(s, r_0)| \geq 2n/3 \).

Define a region-growing weight function \( W(r) \) differentially by

\[
W(0) = 0 \\
W'(0) = c(\delta(B(s, r)))
\]

\( W(r) \) is nondecreasing, bounded below by 0, and bounded above \( \OPT \) (by a similar calculation as in multicut). Now, we have

\[
1 \geq \frac{W(1) - W(0)}{\OPT} = \int_0^{1-r_0} \frac{c(\delta(B(s, r_0 + q)))}{\OPT} dq \\
\geq \frac{cn}{3} \int_0^{1-r_0} |V \setminus B(s, r_0 + q)| dq \overset{(b)}{=} \frac{cn}{3} \sum_{v \in V \setminus B(s, r_0)} d(v, s) - r_0 \\
\geq \frac{cn}{3} \sum_{v \in V} (d(v, s) - r_0) = \frac{cn}{3} \sum_{v \in V} d(v, s) - \frac{cnr_0}{3}. \tag{11.4}
\]
(b) implicitly interchanges changes. In (c), the inequality holds because the sum is extended only by nonpositive terms.

We also have

\[ 1 \leq (d) \sum_{\{u,v\}} d(u,v) \leq (e) \sum_{\{u,v\}} d(u,s) + d(s,v) < n \sum_v d(v,s), \]

where (d) is by the distance constraint in the LP and (e) is by the triangle inequality.

Plugging back into (11.4), we have \(1 > c/6\). For \(c \geq 6\), then, we have a contradiction. \(\square\)

### 11.2.2 Non-concentrated case.

Now we address the remaining non-concentrated case.

**Lemma 11.2.** Suppose \(|B(s,r_0)| \leq 2n/3\) for all \(s \in V\). Then one can compute, a cut with sparsity \(O(\log n) \text{OPT}_L\).

**Proof.** We will apply a region growing technique (similar to multicut) to partition \(V\) into sets \(S_1, \ldots, S_k \subset V\) such that

1. \(c(\bigcup_i \delta(S_i)) \leq O(n^2 \log n) \text{OPT}_L\).
2. \(|S_i| \leq 2n/3\) for all \(i\).

Assuming this holds, then, we obtain the desired sparse cut as follows. Since \(|S_i| \leq 2n/3\) for all \(i\), we can group the \(S_i\)'s into two sets \(T \subset V\) and \(\bar{T} = V \setminus T\) such that \(n/3 \leq |T|, |\bar{T}| \leq 2n/3\). Then \(T\) has sparsity

\[ \frac{c(\delta(T))}{|T||T|} \leq \frac{c(\bigcup_i \delta(S_i))}{(2n/3)(n/3)} \leq O(\log n) \text{OPT}_L. \]

It remains to obtain the sets \(S_1, \ldots, S_k\). As alluded to above, we will use region growing to iteratively remove sets of the form \(B(s,r)\); the key is to show that there are always “good” radii \(r\).

Fix a center vertex \(s \in V\). Consider the region-growing weight function \(W(r)\) defined differentially by

\[
W(0) = \frac{\text{OPT}_L}{n} \\
W''(r) = c(\delta(B(s,r))).
\]
11. Sparsest cut

11.2. Rounding via region growing (for uniform demands)

Kent Quanrud
Fall 2022

$W(r)$ is nondecreasing in $r$ and bounded by the range $[\text{OPT}_L/n, (1 + 1/n) \text{OPT}_L]$. Consequently, for a sufficiently large constant $c_0$, there always exists a radius $r < r_0$ such that

$$W'(0) \leq c_0 n^2 \log(n) W(r),$$

since otherwise the differential inequality $W'(0) \geq c_0 \log(n) n^2 W(r)$ implies that

$$W(r_0) \geq e^{c_0 \log(n)} \text{OPT}_L > (1 + 1/n) \text{OPT}_L,$$

a contradiction. Thus there exists a radius $r \leq r_0$ that, in particular, satisfies

$$c(\delta(B(s,r))) = W'(r) \leq O(n^2 \log(n)) W(r)$$

$$\leq O(n^2 \log(n)) \left(\text{OPT}_L/n + \sum_{v \in B(s,r)} \sum_{e \in \delta(v)} y_e c_e \right).$$

Moreover, by assumption, we have $|B(s,r)| \leq 2n/3$.

We repeat the following steps until $V$ is empty. Here we let $i$ be the index of the iteration.

1. Pick $s_i \in V$ arbitrarily.
2. Find $r_i \in [0, r_0]$ such that

$$c(\delta(B(s,r))) \leq O(n^2 \log(n)) \left(\frac{\text{OPT}_L}{n} + \sum_{v \in B(s,r)} \sum_{e \in \delta(v)} c(e)y(e) \right).$$

3. Set $S_i = B(s, r)$ and $V = V \setminus S_i$.

We claim that these steps produce sets $S_1, \ldots, S_k$ satisfying the two requirements above.

Clearly we have $|S_i| \leq 2n/3$. We also have

$$c\left(\bigcup_i \delta(S_i)\right) \leq \sum_i c(\delta(S_i))$$

$$\leq O(n^2 \log(n)) \left(\text{OPT}_L + \sum_i \sum_{v \in S_i} \sum_{e \in \delta(v)} c(e)y(e) \right)$$

$$= O(n^2 \log(n)) \left(\text{OPT}_L + \sum_{v \in V} \sum_{e \in \delta(v)} c(e)y(e) \right)$$

$$= O(n^2 \log(n)) \text{OPT}_L,$$

which completes the proof. $\square$
11.2.3 Putting it all together.

**Theorem 11.3 (leighton-rao).** The region-growing algorithm described above gives a $O(\log n)$-approximation to the uniform sparsest cut.

*Proof.* Let $y$ be an optimum solution to (11.2). It is easy to identify if $y$ falls in the concentrated or non-concentrated case. For the former, we obtain a $O(1)$-approximation; for the latter, a $O(\log n)$-approximation. \hfill \square

11.3 Rounding via $L_1$-metric embeddings

We now analyze a different 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 (11.1). 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 $d(u,v) = |f(u) - f(v)|$ 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

$$P[e \in \delta(S)] = |f(u) - f(v)| = d(u,v).$$

Thus we can rewrite the sparsity of $d$ as

$$(\text{sparsity of } d) = \frac{\sum_{(u,v) \in E} c(u,v)d(u,v)}{\sum_{(s,t)} d(s,t)b(s,t)} = \frac{E\left[\sum_{e \in \delta(S)} c(e)\right]}{E\left[\sum_{s \in S, t \in \bar{S}} b(s,t)\right]}.$$ (11.5)

Note that the RHS is not the expected sparsity of $S$. The expected sparsity of $S$ is the quantity

$$E[\text{sparsity of } S] = E\left[\frac{\sum_{e \in \delta(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s,t)}\right].$$
is not the same as the quantities in (11.5). 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 $S_1, \ldots, S_{n-1}$ where $\emptyset \subsetneq S_1 \subset S_2 \subset S_3 \cdots \subset S_{n-1} \subsetneq V$. For each $i$, let $p_i = P[S = S_i]$. Then

$$(11.5) = \frac{\sum_{i=1}^{n-1} p_i \sum_{e \in \delta(S_i)} c(e)}{\sum_{i=1}^{n-1} p_i \sum_{s \in S_i, t \in \bar{S}_i} b(s,t)}.$$ 

Now we apply the following elementary fact. (The proof is left as exercise 11.1.)

**Lemma 11.4.** Let $a_1, \ldots, a_h, b_1, \ldots, b_h > 0$. Then

$$\min_i \frac{a_i}{b_i} \leq \sum_i \frac{a_i}{b_i} \leq \max_i \frac{a_i}{b_i}.$$ 

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 $d : (V, 2) \to \mathbb{R}$ 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 $f_1, \ldots, f_h$.
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 than $d$. From the line embedding $f_j : V \rightarrow \mathbb{R}$, we can extract a cut with sparsity at most that of $f_j$. This establishes the following.

**Lemma 11.6.** Let $d : \binom{V}{2} \rightarrow \mathbb{R}$ 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 $\alpha$ of the sparsest metric $d$, then we can covert that into an $\alpha$-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 $f : V \rightarrow \mathbb{R}^h$ (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 $d : \binom{V}{2} \rightarrow \mathbb{R}_{\geq 0}$ be a metric and $\delta \in (0, 1)$. For $h = O(\log(n) \log(1/\delta))$, one can construct a randomized embedding $f : V \rightarrow \mathbb{R}^h$ 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

$$P[\|f(u) - f(v)\|_1 \leq d(u, v)] \leq \delta \quad (11.6)$$

For $\delta = 1/\text{poly}(n)$ and $h = O\left(\log^2 n\right)$, we can apply the union bound to (11.6) 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 (11.1) 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 (11.1)).

This algorithm is due to llr, and establishes the following.

Theorem 11.9. There is a \(O(\log n)\) randomized approximation to (non-uniform) sparsest cut (on undirected graphs).

Actually, llr 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 ??, and left as exercise 11.2.

Now we describe the randomized algorithm of llr that computes the embedding in theorem 11.14. We note that previously bourgain had obtained a deterministic embedding but the output dimension \(h\) was exponential. llr’s algorithm [llr] can be interpreted as an efficient randomized implementation of bourgain’s embedding [bourgain].

The algorithm, which we call random-Fréchet, is extremely simple. We generate \(\lceil \log n \rceil\) coordinates. For \(i = 1, \ldots, n\), we sample a set \(S_i\) where each point is sampled
random-Fréchet\((d : V \times V \to \mathbb{R}_{\geq 0})\)

1. for \(i = 1, \ldots, \lceil \log n \rceil\)
   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 \(f : V \to [0, \log n]\)

Figure 11.1: A \(O(\log n)\) dimension, randomized Frechét embedding with \(O(\log n)\) distortion in expectation

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 \(i\)th 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-Frèchet 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 \(i\)th 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)|.

\[
\begin{align*}
  d(s, u) - d(s, v) &\leq d(u, v) \quad \text{and} \quad d(s, v) - d(s, u) \leq d(u, v) \\
\end{align*}
\]

for all \(s \in S_i\).

**Proof.** By the triangle inequality, we have both

\[
\begin{align*}
  d(s, u) - d(s, v) &\leq d(u, v) \quad \text{and} \quad d(s, v) - d(s, u) \leq d(u, v). \\
\end{align*}
\]

for all \(s \in S_i\). \(\square\)
lemma 11.11 immediate implies the $\|u - v\|_1 \leq O(\log n)d(u, v)$, since there are $O(\log n)$ dimensions and each can contribute at most $d(u, v)$. The harder part is showing the lower bound: informally, we want to show that $\|u - v\|_1 \geq d(u, v)$, 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 $E[\|u - v\|_1] \geq cd(u, v)$ 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.,

$$r_i = \arg\min_{r > 0} \left\{ |\{x : d(u, x) \leq r\}| \geq 2^i, |\{x : d(v, x) \leq r\}| \geq 2^i \right\}$$

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

\[
E[\|u - v\|_1] \geq \sum_{i=0}^{k} E[|u_{i+1} - v_{i+1}|]
\]

\[
\geq c \sum_{i=0}^{k} (\min\{r_i, \delta/2\} - \min\{r_{i-1}, \delta/2\})
\]

\[
\geq \frac{c\delta}{4},
\]

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

\[
E[|u_{i+1} - v_{i+1}|] \geq c(r_i - r_{i-1}),
\]

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-Fréchet embeds \( V \) into \( \mathbb{R}^{\lceil \log n \rceil} \) such that

\[
\|u - v\|_1 \leq \lceil \log n \rceil d(u, v) \quad \text{and} \quad E[\|u - v\|_1] \geq cd(u, v)
\]

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 \( 1/\text{poly}(n) \), the average of \( O(\log n) \) embeddings produced by randomized-Fréchet is an embedding \( V \) into \( \mathbb{R}^{O(\log^2 n)} \) such that

\[
cd(u, v) \leq \|u - v\|_1 \leq C \log(n) d(u, v)
\]

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 randomized-Fréchet) 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 Expander lower bounds

Recall that the integrality gap of an LP relaxation is the maximum (multiplicative) difference between the value of the LP and the discrete problem. The rounding algorithms above show that the integrality gap of the metric LP relaxation (11.1) is at most $O(\log n)$ for uniform sparsest cut (or $O(\log k)$ for $k$ commodity pairs). We want to find a lower bound on the integrality gap as tight as possible.

Consider sparsest cut. The LP asks for the sparsest metric. The sparsest cut is at most a $O(\log n)$-multiplicative factor greater than the sparsity of the sparsest metric. We want to find a family of graphs where the sparsest cut is at least a $\Omega(\log n)$-multiplicative factor greater than the sparsest metric. We will show this for a family of graphs called constant-degree expanders.

An unweighted, undirected graph $G$ is a $\phi$-expander if for all sets $S \subset V$,

$$|\delta(S)| \geq \phi \min\{|S|, |\overline{S}|\}.$$

The definition extends to weighted graphs by replacing the LHS with the weight of the cut. We only need to consider unweighted graphs for our lower bound.

An obvious example of a graph with high expansion is a clique, i.e., a complete graph. The clique on $n$ vertices has expansion $n/2$. An obvious example of a graph with low expansion is a tree; a tree on $n$ vertices has expansion $\Omega(1/n)$. As noted in the introduction, expansion and uniform sparsity are, up to scaling, within a constant factor of one another. Sparse cuts imply low expansion and high expansion imply no sparse cuts.

The clique and tree examples reflects a general intuition that high expansion is correlated with many edges in the graph. But in fact this is not necessarily so: there exists graphs with constant expansion (much higher than a tree) but constant maximum degree (like a tree, and unlike a clique).

**Theorem 11.15.** There exists constants $c,d$ such that for all sufficiently large $n$, there exists a $c$-expander with $n$ vertices and maximum degree $d$.

The fact that a very small (constant degree) graph can be strongly connected (i.e., have large expansion) has vast and profound consequences. We will see this phenomena realized in a few different places in this course. Below we will prove theorem 11.15 by showing that the union a constant number of uniformly random, perfect matchings is an expander with nonzero probability. First, we use theorem 11.15 to lower bound the integrality gap.
11.4.1 Integrality gap for uniform sparsest cut

**Theorem 11.16.** Integrality gap for uniform sparsest cut is $O(\log n)$.

**Proof.** By theorem 11.15, let $G$ be an $n$-vertex $c$-expander with maximum degree $d$, where $c$ and $d$ are universal constants. For every set of vertices $S$, we have

$$(\text{sparsity of } \delta(S)) = \frac{|\delta(|S|)|}{|S||\bar{S}|} \geq \frac{c \min\{|S|, |\bar{S}|\}}{|S||\bar{S}|} \geq \frac{c}{n}$$

where (a) invokes the definition of a $c$-expander.

By LP duality, the sparsest metric has sparsity equal to the maximum throughput $\lambda$ of any concurrent flow. We claim that any concurrent flow has throughput $\lambda$ at most $O(1/n \log n)$. Then multiplicative difference between $c/n$ and $O(1/n \log n)$ is $\Omega(\log n)$, as desired.

To upper bound $\lambda$, we argue that, since $G$ has constant maximum degree, for almost every $(s,t)$-pair, the length of the shortest $(s,t)$-path is at least (roughly) $\log n$. This implies that all the flow paths in the concurrent flow have at least $\log n$, and almost every $(s,t)$-pair requires a total capacity of $\lambda$. Meanwhile the total capacity of the graph is limited because $G$ is sparse, which leads to an upper bound on $\lambda$.

To this end, fix $s$. Since $G$ has maximum degree $d$, there are at most $d^2$ vertices at distance 1 from $s$, at most $d^2$ vertices at distance 2 from $s$, and so forth. In general there are $O(d^i)$ vertices at distance $i$ or less from $s$. For $h = \log_d(n)/2$, there are at most $O(\sqrt{n})$ vertices at distance $h$ or less from $i$. This leaves $n - O(\sqrt{n})$ vertices at distance $h = \Omega(\log n)$ or greater from $s$. Over all $s$, there are $(1-o(1))\binom{n}{2}$ pairs $(s,t)$ at distance $\Omega(\log n)$.

It follows that any concurrent flow with through $\lambda$ – which, in particular, sends $\lambda$ units of flow between $\Omega(n^2)$ $(s,t)$-pairs at distance $\Omega(\log n)$ – requires a total capacity of $O(n^2 \log(n) \lambda)$. But the total capacity in the constant degree expander is $O(n)$. So $\lambda \leq O(1/n \log n)$.

**Remark 11.17.** The proof above used to constant maximum degree to argue that most $(s,t)$-pairs are at distance $\Omega(\log n)$. Meanwhile one can show that a graph with constant expansion has diameter $O(\log n)$\(^1\). See exercise 11.6.

11.4.2 A few random matchings make an expander

We now turn to proving theorem 11.15. While a proof of existence would suffice, we will prove theorem 11.15 via a very simple, randomized construction (for $n$ even):

\(^1\)The diameter of a graph is the maximum $(s,t)$-distance among all $(s,t)$ pairs.
namely, for a fixed set of \( n \) vertices, the union of a few (independently sampled) uniformly random perfect matchings over these \( n \) vertices has constant expansion. We remark that there are many other constructions and in particular there is interest in questions such as deterministic constructions, implicit constructions of very largeexpanders with compact oracle access, and extreme families such as Ramanujan graphs.

**Theorem 11.18.** Let \( n \) be even and sufficiently large. There exists constants \( c, d > 0 \) such that the union of \( d \) independent and uniformly random matchings of a set of \( n \) vertices form a \( c \)-expander with high probability.

**Proof.** Let \( d \in \mathbb{N} \) be a parameter to be determined, and consider the union of \( d \) perfect matchings over a fixed set of \( n \) vertices. We want to show that for every set \( S \) of size \( |S| \leq n/2 \), \( |\delta(S)| \geq c|S| \) for some constant \( c \).

We reframe this claim as follows. Let \( S \subseteq V \) be of size \( k \overset{\text{def}}{=} |S| \leq n/2 \). Let \( T \subseteq V \setminus S \) be of size \( |T| = k/6 \).

**Claim.** With nonzero probability, for all such \( S \) and \( T \), some vertex in \( S \) is matched to a vertex outside \( S \cup T \).

The claim directly implies that, for all sets \( S \) with at most \( n/2 \) vertices, \( |N(S) \cup S| \geq (7/6)|S| \). (Otherwise take \( T = N(S) \setminus S \).) In turn, for all such \( S \), we have

\[
|\delta(S)| \geq |N(S) \cup S| - |S| \geq |S|/6,
\]
as desired.

We will prove the 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 for generating a perfect matching.

1. Index the vertices \( v_1, \ldots, v_n \) such that \( S = \{v_1, \ldots, v_k\} \) and \( T = \{v_{k+1}, \ldots, v_{7k/6}\} \).

2. Repeat \( n/2 \) times:
   (a) Let \( v_i \) be the unmatched vertex of smallest index
   (b) Sample \( v_j \) among the remaining unmatched vertices uniformly at random.
   (c) Match \( v_i \) with \( v_j \)

While this procedure might appear biased by \( S \) and \( T \), we argue that it is not.
Claim. The above steps generate a uniformly random perfect matching.

Indeed, observe that each random choice in the procedure leads to a different matching. There are \((n - 1)(n - 3)\cdots 1\) possible outcomes and all are equally likely. There are also \((n - 1)(n - 3)\cdots 1\) possible matchings. This proves the claim.

Thus we may assume that our \(d\) perfect matchings are generated by the procedure above. Now, for a single matching, the probability that \(S\) is matched to \(\bar{S}\) is bounded above by probability that first \(k/2\) choices of \(v_j\)'s are in \(S\cup T\). The probability of this is bounded above by

\[
\prod_{i=1}^{k/2} \frac{(7/6)k - 2i + 1}{n - 2i + 1} \leq \prod_{i=k/3+1}^{k/2} \left( \frac{k/2}{n/2} \right) = \left( \frac{k}{n} \right)^{k/6} \leq \left( \frac{n}{k} \right)^{-1/6}.
\]

In (a) we observe that

\[(7/6)k - 2i + 1 \leq k/2 \text{ iff } i \geq k/3 + 1/2\]

and that

\[n - 2i + 1 \geq n/2 \text{ iff } i \leq n/4 + 1/2, \text{ if } i \leq k/2 + 1/2\]

For \(d\) matchings, the probability that \(S\) is matched to \((S \cup T)\) in all \(d\) matchings is bounded above by \(\left( \frac{n}{k} \right)^{-d/6}\) Now we take a union bound over all choices of \(S\) and \(T\). We have

\[
\left( \frac{n}{7k/6} \right) \left( \frac{7k/6}{k} \right) \left( \frac{n}{k} \right)^{-d/6} = \left( \frac{n}{k} \right)^{1-d/6} \leq \left( \frac{n}{k} \right)^{-\Omega(d)}
\]

for a sufficiently large constant \(d\). Taking a union bound over \(k = 1, 2, \ldots, n/2\), we conclude that with high probability, all sets \(S\) with \(|S| \leq n/2\) have at least \(|S|/6\) neighbors. So the union of a constant number of random matchings is a \((1/6)\)-expander.

11.5 Application: Minimum bisection

A bisection is a partition of the vertices \(V\) into \((S, \bar{S})\) of (essentially) equal size: \(|n/2| \leq |S|, |\bar{S}| \leq |n/2|\). 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 \([n/2]\) vertices each. In any case the minimum bisection problem is to compute a vertex set \(S \subset V\) of size \(|S| = [n/2]|\) 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 11.3 guides the reader through a formal proof of the argument.

**Theorem 11.19.** 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.
11.6 Exercises

Exercise 11.1. Prove lemma 11.4.

Exercise 11.2. Recall the randomized $O(\log n)$ approximation algorithm for sparsest cut based on $L_1$-embeddings. Show how to adjust the algorithm and analysis to obtain a randomized $O(\log k)$ approximation factor where $k$ is the number of commodities with nonzero demand.

Exercise 11.3. Recall the bicriteria approximation algorithm for the minimum bisection problem from section 11.5.

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

\[
\frac{c(\delta(S_i))}{|S_i|} \leq O(\log(n)) \frac{\text{OPT}}{n}.
\]

3. Combine the two parts above to prove that the algorithm returns a $1/3$-balanced cut of size $O(\log n) \cdot \text{OPT}$.

Exercise 11.4. One generalization of uniform sparsest cut is to directed graphs. Here one is given a directed graph $G$ with edge costs $c$; the (directed, out-) sparsity of a set $S$ is defined

\[
\frac{c(\delta^+(S))}{|S||\bar{S}|}
\]

where we recall that $\delta^+(S)$ is the directed cut of edges leaving $S$. The directed uniform sparsest cut problem to find the set $S$ minimized the directed sparsity just defined. Extend leighton-rao’s algorithm \[leighton-rao\] to obtain a $O(\log n)$ approximation for the directed uniform sparsest cut.\[2\]

Exercise 11.5. 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 exercise 11.4). 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.

\[2\]This is not an easy exercise.
Exercise 11.6. Prove that any graph $G$ with constant expansion has diameter $O(\log n)$.

11.7 Additional notes and references

One can do better than a $O(\log n)$ approximation for uniform sparsest cut – arv gave a $O(\sqrt{\log n})$ 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-arv-notes.

We refer to spielman; trevisan-spectral-notes for further background on expanders, especially in the context of spectral graph theory. ?? follows the proof in [trevisan-spectral-notes].
Chapter 12

Tree Metrics

12.1 Introduction

All of our problems involve paths in graphs, whether packing paths in flow, cutting paths in cuts, or evaluating shortest paths in fractional cuts. 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. Any of our graph problems would have been trivial in a tree. Here we will study a bold approach to graph algorithms based on this idea: processing the input graph $G$ to produce a tree $T$ that somehow reflects $G$ and preserves its salient properties, solve the problem on $T$, and transfer 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 $G$, $d_G$ 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 $e = \{u,v\}$, we say that 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) $\alpha$, for $\alpha \geq 1$, if every edge has stretch at most $\alpha$. 
A natural goal is to obtain a spanning tree with small uniform stretch, but 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 ?? 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.** Akpw 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-96; Bartal-98 ignored the requirement that $T$ is a spanning tree of $G$; more general he sought auxiliary trees $T$ where the vertex of $G$ correspond to the leaves of $G$, while retaining the property $T$ that $d_T \geq d_G$. He produced randomized trees where for each edge the average stretch was polylog($n$):

$$E_T[(\text{stretch } e)] \leq O(\text{polylog}(n))$$

Note that this a different sense of “average stretch” then above. We will present an algorithm of [frt] building on [bartal-98] 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 [akpw]. Here we recall that we want to compute a spanning tree $T$.

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 edge cut edge $e$ has stretch $O(R)$ in $T'$, but this expands out to stretch $O(R^2)$ with
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 $C_v$ of all vertices (remaining) in $V$ within some distance $R' \leq R$ 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.

respect to $T$ because passing through a vertex in $T'$ actually corresponds to traversing a path of length $O(R)$ in the underlying tree.

So we have a problem where each step of the recursion induces an additional factor of $R$. 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 external, or cut, edges? Region growing! Low diameter decompositions!

If we partition the graph by region growing techniques, we can ensure that

$$(# \text{ external edges}) \leq \frac{O(\log m)}{R}m.$$ 

This revises the recursive bound as

$$f(m) \leq Dm + (D + 1)f(c \log(m)m/D)$$
for a constant $c > 0$. For $D = e^{\sqrt{\frac{\log(m)}{\log \log m}}}$ the recursion is bounded by $f(m) = e^{O\left(\sqrt{\frac{\log m}{\log \log m}}\right)}$. (The calculations are given below.)

**Theorem 12.1.** The AKPW algorithm returns a spanning tree with average stretch $e^{O\left(\frac{\log m}{\log \log m}\right)}$.

**Solving the recursion.** 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

$$h = O\left(\log_D/c_0 \log(m) \right) = O\left(\log(m)/\log(D)\right) = O\left(\frac{\log(m)}{\log(D)}\right)$$

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

$$f(m) \leq me^{O\left(\sqrt{\frac{\log(m) \log \log(m)}}\right)},$$

as desired.
12.3 Hierarchical Tree Metrics

We now discuss...

The [akpw] algorithm was able to obtain low stretch \textit{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 when we allow for randomization. Perhaps we can output a \textit{randomized} tree \(T\), such that for each edge \(e\), the expected value of \(e\)'s stretch is \(o(n)\). Such a claim does \textit{not} contradict the lower bound for the \(n\)-vertex cycle; in fact, one can get constant stretch for the cycle which we leave as ??.

To build some intuition, let us start from the [akpw] 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 [akpw].

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 produced a randomized \textit{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 \(\alpha^i\) for a fixed constant \(\alpha\). 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 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 [frt]'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) \(\alpha\) and ordering for every \(i\).
intersections of these balls (across $i$'s) induce a laminar family of sets over $V$ which are arranged as a tree.

[frt]'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.** [frt]'s algorithm produces a randomized hierarchical tree $T$ such that for each edge $e$, $E[\text{stretch}_e] \leq O(\log n)$.

To prove the theorem, fix an edge $e = \{u, v\}$. Recall that $d(u, v)$ is decided, up to a constant factor, by the height $k$ where $u$ and $v$ are first separated. (Then $d(u, v) = O(4^k)$.) 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

$$d_T(u, v) \leq \sum_w O(1) (\text{contrib. of } w \text{ to } d_T(u, v)).$$ (12.1)
Now, fix a vertex $w$. Suppose that $w$ was the $\ell$th 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

$$E[\text{contrib. from } w \text{ to } d_T(u, v)] \leq O(1/\ell) d(u, v).$$

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 $\ell$th closest vertex to $u$ or $v$. Then

$$E[\text{contrib. from } w \text{ to } d_T(u, v)] \leq 4d(u, v)/\ell.$$

**Proof.** We first observe that $w$ contributes 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 if 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 $\alpha$ 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

$$E[\text{contrib. from } w \text{ to } d_T(u, v)] \leq 2.5d(u, v) P[E_2] = 2.5d(u, v)/\ell,$$

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

$$d(w, u) \geq d(w, v) - d(u, v) \geq 4^k - 4^k/2 > 24^{k-1}$$

180
which rules out smaller values of $k$. Here (a) is by the the triangle inequality and (b) is by assumption on $\alpha$. 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

$$\Pr[E_1] \leq \frac{|[d(w, u), d(w, v)]|}{|[4^k, 24^k]|} \leq \frac{d(u, v)}{4^k}$$

by (c) the triangle inequality. Thus

$$E[\text{contrib. from } w \ldots] \leq 4^{k+1} \Pr[E_1] \Pr[E_2] \leq 4d(u, v)/\ell,$$

as desired. \hfill \qed

12.4 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.** Recall that the low-stretch spanning tree of [AKPW] 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 $\text{poly}(n)$. 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 12.3.** Show how to use the randomized tree metric to randomly round the sparsest cut LP and obtain $O(\log n)$-approximation for sparsest cut.

**Exercise 12.4.** Prove that the $O(\log n)$ bound is tight for tree metrics (up to constants).

**Exercise 12.5.** The randomized tree metric produces a distribution such that each edge has randomized stretch that is $O(\log n)$ in expectation. Design an algorithm that produces a distribution over spanning trees such that each edge has stretch $n^{o(1)}$ in expectation.\(^1\)

\(^1\)Hint: Consider the techniques of ??.
Chapter 13

Two Theorems by Claude Shannon

13.1 Introduction

Today we will discuss two papers by Claude Shannon published in the Bell Systems Technical Journal at roughly the same time.

- shannon-coding
- shannon-circuits

While we will not discuss it today, we point out that Shannon another important paper at the same time, this time laying foundations for cryptography.

- shannon-crypto

13.2 Circuit lower bounds

The first result of Shannon that we will discuss is about circuits, and in particular, lower bounds for circuits. This result was not Shannon’s first famous encounter with circuits. His 1936 Master’s thesis, born out of his experience as a research assistant working on Vannevar Bush’s differential analyze, identified the logical equivalence between the switching circuits (that the differential analyzer was composed of) and boolean algebra.

The current discussion on circuit fast forwards a little more than ten years for Shannon’s masters thesis. It came at an exciting time for Bell Labs: in late 1947, Bardeen, Brattain, and Shockley in the Solid States Physics group invented the first transistor. This sets off the digital revolution, as transistors rapidly improve and coalesce into the CPU’s we know today. For the sake of this discussion, we can think of transistors as extremely fast and efficient physical circuits.
13. Two Theorems by Claude Shannon

13.2. Circuit lower bounds

For us, a circuit is a family of logical gates arranged in a directed acyclic graph. We assume the reader has had some acquaintance with gates before (see, e.g., [arora-barak; sipser]), and we briefly review. An example of a circuit is given in fig. 13.1. We have four types of gates. One is an input gate, representing a bit from the input. In the picture above, we have input gates for 8 bits, \( x_1, \ldots, x_8 \). We then have logical gates, that take as input a finite number of bits and output a bit based on applying either an “and”, \( \land \), an “or”, \( \lor \); or a negation, \( \neg \). The \( \neg \) takes only one input bit and the other two logical gates can take any finite number of bits. The input bits of a gate come from the outputs of some other gates. If we draw a directed edge from one gate to another based on the flow of information, this graph must form a DAG.

Theorem 13.1. For all \( n \in \mathbb{N} \), there exists a function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) that cannot be computed by any circuit of size \( < 2^n/cn \) for some universal constant \( c > 0 \).

Proof. We first count the number of circuits of some size. We first observe that any circuit of size \( k \) can be encoded in \( ck \log(k) \) bits, for some universal constant \( c > 0 \), by interpreting a circuit of size \( k \) as a graph with \( k \) vertices and edges. Each edge and each vertex requires \( O(\log(k)) \) bits to encode. In turn, there are at most \( 2^{ck \log(k)} \) different circuits of size \( k \). For \( k = 2^n/cn \), this is less than \( 2^n \).

Now, let \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) be a uniformly random boolean function. For each circuit \( C \) of size \( \leq 2^n/cn \), the probability that \( f(x) = C(x) \) for all \( x \in \{0, 1\}^n \) is \( < 1/2^n \). Thus the expected number of circuits of size \( \leq 2^n/cn \) that agree with \( f \) on all inputs is \( < 1 \). By the probabilistic method, there exists an \( f \) that is unequal to all circuits of size \( \leq 2^n/cn \). \( \square \)
13.2.1 A Circuit Hierarchy Theorem

An important application is the following Hierarchy theorem for circuits.

**Definition 13.2.** Let $f : \mathbb{N} \to \mathbb{N}$ be a function. A circuit family of size $f$ is a sequence of boolean circuits $C_1, C_2, \ldots$, where $C_n : \{0, 1\}^n \to \{0, 1\}$ and $|C_n| \leq f(n)$.

**Definition 13.3.** A language $L \subset \{0, 1\}^*$ has circuit complexity $f$, denoted $L \in \text{CircuitSize}(f)$, if there is a circuit family of size $f$, $\{C_i : \{0, 1\}^i \to \{0, 1\}\}$, such that for each $x \in \{0, 1\}^n$, whether $x \in L$ can be decided by the circuit $C_n$.

Some basic observations about circuits.

1. Every polynomial time language has polynomial circuit complexity.
2. Circuit Satisfiability and Boolean Satisfiability are polynomial time equivalent.
3. The first two points combine to give a proof of NP-Completeness [cook; levin].

Given the importance of circuits, it is important to understand what circuits of a given size can and cannot do. Shannon’s theorem above gives some limits to the power of circuits. The following Hierarchy theorem, the proof of which uses Shannon’s theorem, shows that the size of a circuit matters at a fine grained level: there always exists boolean functions that a larger size can implement, but a smaller size cannot.

**Theorem 13.4.** There exists a universal constant $c > 0$ for which the following holds. Let $f : \mathbb{N} \to \mathbb{N}$ with $n \leq f(n) \leq 2^n/cn$. Then

$$\text{CircuitSize}(f) \subsetneq \text{CircuitSize}(cf).$$

**Proof.** Here we will prove a weaker bound with $cf \log f$ instead of $f$. Fix $n \in \mathbb{N}$ and let $m = f(n)$. We show that there is a boolean function $g : \{0, 1\}^n \to \{0, 1\}$, such that $f$ has a $O(m \log m)$-size circuit but not an $m$-size circuit.

For any $k \in \mathbb{N}$, we know that there is a function $g : \{0, 1\}^k \to \{0, 1\}$ that requires at least $2^k/ck$ size, and at most $ck2^k$ size, for some constant $c > 0$.

We choose $k$ large as possible such that $2^k/ck \leq m$. Then

$$ck2^k \leq (ck)^2(m + 2) \leq O(m \log(m)).$$

\[ \square \]
13.3 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 Shannon. 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 again. Your friend again suggests that they didn’t understand you. So you say it 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 $C : \{0, 1\}^m \to \{0, 1\}^n$ and a decoder $D : \{0, 1\}^n \to \{0, 1\}^m$. The encoder takes the input message from $x \in \{0, 1\}^m$ and maps to a longer message $\{0, 1\}^n$, where $n \geq m$. The encoded message $C(x)$ is transmitted. On the other end, let $N : \{0, 1\}^n \to \{0, 1\}^n$ be a random function that flips each bit independently with probability $p$

\[
\begin{align*}
  x & \xrightarrow{\text{encode}} C(x) \xrightarrow{\text{bits flip w/ prob. } p} N(C(x)) \xrightarrow{\text{decode}} D(N(C(x)))
\end{align*}
\]

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

\[
P[D(N(C(x))) \neq x]
\]

over the randomness in $N$ and over $x \in \{0, 1\}^m$ chosen uniformly at random.

Before introducing Shannon’s theorem for codes, we have to introduce one more character: entropy.
Definition 13.5. Let \( X \in \mathcal{X} \) be a discrete random variable. The entropy of \( X \), denoted \( H(X) \), is defined as

\[
H(X) = \sum_{x \in \mathcal{X}} -P[X = x] \log(P[X = x]),
\]

where 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 \( P[X = 1] = p \). 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.

Theorem 13.6. Consider transmission over a noisy channel where each bit is flipped independently with probability \( p \in (0, 1/2) \).

1. For all \( \delta > 0 \), and \( n \) sufficiently large, there is a coding scheme \((C : \{0, 1\}^m \to \{0, 1\}^n, D : \{0, 1\}^n \to \{0, 1\}^m)\) that has transmission rate \( \geq 1 - H(p) - \epsilon \) and average error rate \( \leq \epsilon \).

2. For all fixed \( \delta > 0 \), and \( n \) sufficiently large any coding scheme \((C : \{0, 1\}^m \to \{0, 1\}^n, D : \{0, 1\}^n \to \{0, 1\}^m)\) with transmission rate \( \geq 1 - H(p) + \delta \) has average error rate \( > 1 - \delta \).

13.3.1 A helpful inequality

Before proving Shannon’s theorem, we mention a very helpful identity that relates entropy to sums of binomial coefficients. Entropy enters the analysis of Shannon’s upper bound via this lemma.

Lemma 13.7. Let \( n \in \mathbb{N} \) and \( p \in (0, 1) \). Then

\[
\sum_{i=0}^{m} \binom{n}{i} \leq 2^{H(p)n}.
\]

We will prove lemma 13.7 below in section 13.4 where we discuss entropy in greater detail.
13.3.2 Shannon’s Upper Bound

In this section, we probe Shannon’s upper bound - the first claim in theorem 13.6. We first state the part that is relevant.

**Theorem 13.8.** For all \( \delta > 0 \), there exists a coding scheme 
\[ (C : \{0,1\}^m \to \{0,1\}^n, D : \{0,1\}^n \to \{0,1\}^m) \]
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 \( m = (1 - H(p) - \delta)n \). Let \( C : \{0,1\}^m \to \{0,1\}^n \) be a uniformly random function. Define \( D : \{0,1\}^n \to \{0,1\}^m \) by setting \( D(y) \) to be the point \( x \) closest to \( C(x) \), breaking ties arbitrarily.

Since \( H(p) \) is continuous, we can choose \( \epsilon > 0 \) sufficiently small such that \( |H((1 + \epsilon)p) - H(p)| \leq \frac{\delta}{2} \).

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 \( x' \in \{0,1\}^m, x' \neq x \) is within \( (1 + \epsilon)pn \) bits of \( C(N(x)) \).

2. For sufficiently large \( n \), with probability of error \( \leq \delta/2 \), the noisy transmission \( N(C(x)) \) flips at most \( (1 + \epsilon)pn \) bits in \( C(x) \).

Suppose the above holds and let \( n \) be sufficiently large. Then with combined probability of error \( \leq \delta \), \( C(x) \) is the only point within \( (1 + \epsilon)pn \) bits of \( C(x) \). That is, when we consider all of the randomness over the random of choice of \( x \in \{0,1\}^m \), the random encoding function \( C : \{0,1\}^m \to \{0,1\}^n \), and the noise \( N(C(x)) \), we have
\[
P_{x,C,N}[D(N(C(x))) \neq x] \leq \delta.
\]

We can rewrite this as
\[
E[C][average error of C] = E[C] \left[ P_{x,C,N}[D(N(C(x))) \neq x] \right] \leq \delta.
\]

To dramatically complete the proof: by the probabilistic method, there exists an encoding \( C : \{0,1\}^m \to \{0,1\}^n \) 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 $N(C(x))$.

We prove the claim conditional on $y = N(C(x))$; the unconditional claim immediately follows. Fix $y = N(C(x))$. Consider any other input point $x' \in \{0,1\}^m$. By construction, $C(x')$ is selected uniformly at random from $\{0,1\}^n$. Therefore

$$P[\|C(x') - C(x)\|_0 \leq (1 + \epsilon)pn] = 2^{-n} \sum_{i=0}^{(1+\epsilon)pn} \binom{n}{i} \leq 2^{(H((1+\epsilon)p)-1)n}.$$ 

By the union bound, we have

$$P[\|C(x') - C(x)\|_0 \leq (1 + \epsilon)pn \text{ for some } x' \neq x] \leq 2^{m+(H((1+\epsilon)p)-1)n}.$$

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 $\epsilon$, we have

$$1 - H((1 + \epsilon)p) - O\left(\frac{\log(1/\epsilon)}{n}\right) \geq 1 - H(p) - \delta/2 - O\left(\frac{\log(1/\epsilon)}{n}\right) \geq 1 - H(p) - \delta$$

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 $N(y)$ flips at most $(1 + \epsilon)pn$ bits in $y$.

We have

$$\lim_{n \to \infty} P[(\text{# bits flipped}) \geq (1 + \epsilon)pn] \overset{(a)}{=} \lim_{n \to \infty} e^{-e^2pn/2} = 0,$$

where (a) applies the Chernoff inequality. \hfill \Box

13.3.3 Lower bounds for Shannon’s capacity

Theorem 13.9. Let $\delta > 0$. For $n$ sufficiently large, no coding scheme $(C : \{0,1\}^m \to \{0,1\}^n, D : \{0,1\}^n \to \{0,1\}^m)$ can have transmission rate
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 \((1 - \epsilon)p_n\) bits from \(\mathcal{C}(x)\) and no more than \((1 + \epsilon)p_n\) bits from \(\mathcal{C}(x)\). Let \(\bar{E}\) be the complementary event.

We have

\[(\text{average correctness rate}) \leq \mathbb{P}[\bar{E}] + \mathbb{P}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) = x | E].\]

For the first term, we have

\[\mathbb{P}[\bar{E}] = \mathbb{P}[\|\mathcal{N}(\mathcal{C}(x)) - \mathcal{C}(x)\|_0 < (1 - \epsilon)p_n] + \mathbb{P}[\|\mathcal{N}(\mathcal{C}(x)) - \mathcal{C}(x)\|_0 > (1 + \epsilon)p_n] \leq 2e^{-2pn/3}\]

by Chernoff bounds. In particular, for fixed \(\epsilon > 0\),

\[\lim_{n \to \infty} \mathbb{P}[\bar{E}] = 0.\]

To bound the second term, for each \(x\), let

\[Y_x = \{y \in \mathcal{D}^{-1}(x) : (1 - \epsilon)p_n \leq \|x - y\|_0 \leq (1 + \epsilon)p_n\}.\]

We have

\[\mathbb{P}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) = x | E] = \frac{1}{2^m} \sum_{x \in \{0,1\}^m} \sum_{y \in Y_x} \mathbb{P}[\mathcal{N}(\mathcal{C}(x)) = y].\]

For each \(x\), and for each \(y \in Y_x\), we have

\[\mathbb{P}[\mathcal{N}(\mathcal{C}(x)) = y] \leq p^{(1-\epsilon)p_n} (1-p)^{(1-(1-\epsilon)p)n} = 2^{-H(p)n} \left(1 - \frac{p}{p}\right)^{\epsilon pn}\]

Here (a) is because because \(y\) differs in at least \((1 - \epsilon)p_n\) bits. We now have

\[\mathbb{P}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) = x | E] \leq \frac{1}{2^m} \sum_{x \in \{0,1\}^m} e^{-H(p)n} \left(1 - \frac{p}{p}\right)^{\epsilon pn} |Y_x|\]

\[\leq 2^{(1-H(p))n-m} \left(1 - \frac{p}{p}\right)^{\epsilon pn}\]

\[\leq 2^{-\delta n} \left(1 - \frac{p}{p}\right)^{\epsilon pn} = 2^{\epsilon \log \left(\frac{1-p}{p}\right) - \delta n}.\]

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 \(n \to \infty\). \(\square\)
13.4 Entropy

In this section, we explore some of the many interesting properties of entropy. We restate the definition for the reader’s convenience.

**Definition 13.5.** Let \( X \in \mathcal{X} \) be a discrete random variable. The entropy of \( X \), denoted \( H(X) \), is defined as

\[
H(X) = \sum_{x \in \mathcal{X}} -P[X = x] \log(P[X = x]),
\]

where 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 \( P[X = 1] = p \). 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

\[
H(X) = E_X \left[ \log \left( \frac{1}{P[X' = X]} \right) \right].
\]

Put another way, given a discrete random variable \( X \in \mathcal{X} \), let us define the **shock** of \( X \), \( S_X > 0 \), as the random variable that, if \( X = x \), takes the value

\[
S_X = \frac{1}{P[X = x]}.
\]

Here \( P[X = x] \) refers to the a priori probability of \( X \) equaling \( x \). Then the entropy of \( X \) is

\[
H(X) = E[\log(S_X)].
\]

13.4.1 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

\[
pf(x) + (1 - p)f(y) \leq f(px + (1 - p)y).
\]

\(^1\)This is not a standard terminology.
By induction, we can extend this to finite convex combinations of points. Let \( x_1, \ldots, x_n \in [a, b] \) and \( p_1, \ldots, p_n \geq 0 \) with \( p_1 + \cdots + p_n = 1 \). Then we have

\[
 p_1 f(x_1) + \cdots + p_n f(x_n) \leq f(p_1 x_1 + \cdots + p_n x_n). \tag{13.1}
\]

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 (13.1) is the same as saying that

\[
 E[f(X)] \leq f(E[X]).
\]

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 13.10.** Let \( X \in [a, b] \) be a random variable, and let \( f : [a, b] \to \mathbb{R} \) by concave. Then

\[
 E[f(X)] \leq f(E[X]).
\]

**Proof.** Suppose \( X \) takes on only two values, \( a \) and \( b \), with probability \( p \) and \( 1-p \) respectively. Then

\[
 E[f(X)] = pf(a) + (1-p)f(b) = f(pa + (1-p)b) = f(E[X]).
\]

We can extend the argument to any finite number of values by induction. \( \square \)

**Lemma 13.11.** 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

\[
 H(X) = E[\log(S_X)] \overset{(a)}{\leq} \log(E[S_X]) \overset{(b)}{=} \log(|X|)
\]

(a) is by Jensen’s inequality. (b) is because

\[
 E[S_X] = \sum_x \frac{P[X = x]}{P[X = x]} = n.
\]

On the other hand, if \( X \) is the uniform distribution over \( n \) values \( x_1, \ldots, x_n \), then

\[
 H(x) = \sum_i P[X = x_i] \log\left(\frac{1}{X = x_i}\right) = \sum_i \frac{1}{n} \log(n) = \log(n).
\]

\( \square \)
13.4.2 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 13.12.** The conditional entropy of \(Y\) on \(X\) is defined as

\[
H(Y \mid X) = E_X[H(Y) \mid X]
\]

In terms of “shocks”, we let \(S_{Y \mid X}\) be the shock value of the conditional variable \(Y\) given \(X\). To be precise, conditional on \(X = x\) and \(Y = y\), \(S_{Y \mid X}\) takes the value

\[
S_{Y \mid X} = \frac{1}{P[Y = y \mid X = x]},
\]

where \(P[Y = y \mid X = x]\) is the *a priori* probability given only \(X = x\). Then

\[
H(Y \mid X) = E_X[E_Y[\log(S_{Y \mid X})]] = E_{X,Y}[\log(S_{Y \mid X})]
\]

**Lemma 13.13.**

\[
H(X, Y) = H(Y \mid X) + H(X)
\]

**Proof.** Observe that conditional on \(X = x\) and \(Y = y\), we have

\[
S_{X,Y} = \frac{1}{P[X = x, Y = y]} = \frac{1}{P[X = x]P[Y = y \mid X = x]} = S_X S_{Y \mid X}.
\]

Thus

\[
H(X, Y) = E_{X,Y}[\log(S_{X,Y})] = E_{X,Y}[\log(S_{Y \mid X}) + \log(S_X)] = H(Y \mid X) + H(X),
\]

as desired. \(\square\)

13.5 Principle of Independence

**Lemma 13.14.** Let \((X, Y)\) be jointly distributed. Then \(H(Y \mid X) \leq H(Y)\)

**Proof.** We have

\[
H(Y \mid X) = E_{X,Y}[\log(S_{Y \mid X})] = E_Y[E_{X}[\log(S_{Y \mid X}) \mid Y]]
\]

\[
\leq (a) E_Y[\log(E_{X}[S_{Y \mid X} \mid Y])] \leq H(Y).
\]
13. Two Theorems by Claude Shannon

13.5. Principle of Independence

(a) is by Jensen’s inequality. (b) is because, conditional on $Y = y$, we have

$$
E_X \left[ S_{Y|X} \mid Y \right] = \sum_x \frac{P[X = x \mid Y = y]}{P[Y = y \mid X = x]} \leq \sum_x \frac{p(x)}{p(y)} = \frac{1}{p(y)}.
$$

(c) substitutes Bayes’ law:

$$
P[X = x \mid Y = y] P[Y = y] = P[X = x, Y = y] = P[Y = y \mid X = x] P[X = x].
$$

Maximality of independence

**Lemma 13.15.** Fix marginal probabilities over two finite sets $\mathcal{X}$ and $\mathcal{Y}$. Over all joint distributions of $(X, Y) \in \mathcal{X} \times \mathcal{Y}$, the entropy of $(X, Y)$ is maximized by taking $X$ and $Y$ to be independent.

Subadditivity

**Lemma 13.16.** Let $(X, Y)$ be a joint distribution of discrete random variables. Then

$$
H(X, Y) \leq H(X) + H(Y).
$$

13.5.1 Bounding sums of binomial coefficients

Finally, let us prove lemma 13.7 which we recall was the key to the proof of Shannon’s upper bound.

**Lemma 13.7.** Let $n \in \mathbb{N}$ and $p \in (0, 1)$. Then

$$
\sum_{i=0}^{pn} \binom{n}{i} \leq 2^{H(p)n}.
$$

**Proof.** Let $M$ denote the sum on the LHS. We interpret $M$ as the number of subsets of $[n]$ with $\leq \alpha n$ elements. Define a discrete random variable $X$ as a uniformly random set with $\leq \alpha n$ elements. $X$ has entropy

$$
H(X) = \log(M)
$$

by the maximality principle. We can identify $X$ with the joint distribution $(Y_1, \ldots, Y_n)$, where $Y_i$ indicates whether element $i$ appears in $X$. Each $Y_i$ (taken alone) is a Bernoulli variable with probability $p$. Then

$$
H(X) \leq \sum_{i=1}^{n} H(X_i) = nH(p).
$$

Thus $\log(M) = nH(p)$, as desired. \qed
13.5.2 What is entropy, really?

We close with a quote from Shannon [tm].

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”.

13.6 Exercises

Exercise 13.1. In section 13.3, we discussed coding schemes for achieving low average error rates. Recall that an average error rate of $\delta$ means that the error is averaged over all $x \in \{0, 1\}^m$:

$$\text{average error} = \mathbb{E}_x \left[ P \left[ D(N(C(x))) \neq x \right] \right] = \mathbb{P}_{x \sim N} \left[ D(N(C(x))) \neq x \right].$$

By contrast, a uniform error of $\delta$ means that for every $x \in \{0, 1\}^m$, the probability of transmitting and failing to decode $x$ is at most $\delta$.

$$\text{uniform error} = \max_{x} \mathbb{P}_{N} \left[ D(N(C(x))) \neq x \right].$$

Prove Shannon’s theorem (theorem 13.6) for uniform error instead of average error.

Exercise 13.2. In section 13.3, 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 $\Sigma$ be a finite alphabet of $n$ letters. Our goal is to efficiently assign bit strings (codes) to each letter in $\Sigma$ so that messages, composed of sequences of letters in $\Sigma$, are as efficient as possible. More specifically, we are only allowed to use prefix-free codes, which are mappings

$$C : \Sigma \rightarrow \{0, 1\}^*$$

assigning bit strings (of varying length) to letters such that no code $C(x) \ (x \in \Sigma)$ is a prefix to another code $C(y) \ (y \in \Sigma)$. 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 $\Sigma$ 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 $\Sigma$ are distributed frequently. (This is where we have some opportunity for compression) Let $p \in \Delta^\Sigma$ be a fixed distribution over $\Sigma$. For each letter in $x \in \Sigma$, $p_x$ represents the average frequency of the letter $x$ in these messages.

Given an encoding $C : \Sigma \rightarrow \{0,1\}^*$, the average number of bits per letter is

$$\sum_{x \in \Sigma} p_x |C(x)|,$$

where $|C(x)|$ denotes the length of the bit string $C(x)$.

For this problem, consider a special case where every probability $p_x$ is a power of 2, of the form $\frac{1}{2^{i_x}}$ for some integer $i_x \in \mathbb{N}$. Show that there exists a prefix code $C : \Sigma \rightarrow \{0,1\}^*$ where the average length is exactly $H(p)$, where $H(p)$ is the entropy of the random letter drawn from $\Sigma$ 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.
Chapter 14

A Linear Map of the Web

Figure 14.1: Map of a small part of the world wide web around wikipedia.org [www-map].
14.1 Ranking the web

The world wide web is a large and messy place. As of March 2022, [http://worldwidewebsize.com](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 them 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 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 given bestow some approval. Another appealing aspect 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, wouldn’t that dilute the “approval” bestowed by $u$? As an analogy, suppose there is 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? The next idea 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.
Idea 2. Score each website equal to the sum, over all other webpages linking to it, of the reciprocal of the number of vertices

\[ \text{score}_2(v) = \sum_{u: (u,v) \in E} \frac{1}{\sum_{w: (u,w) \in E} 1}. \]

For ease of notation, let \( d^+(u) \) denote the number of edges leaving a vertex \( u \), a.k.a. the out-degree. Then \( \text{score}(2) \) can be written as

\[ \text{score}_2(v) = \sum_{u: (u,v) \in E} \frac{1}{d^+(v)}. \]

Unfortunately \( \text{score}_2 \) can be manipulated as follows. To promote a website \( v \), one need only create many fake websites \( u \) with a link to \( v \) to drive up \( \text{score}_2(v) \). So perhaps when evaluating a link \( (u,v) \) we need a mechanism to decide if \( u \) is much of an authority to begin with.

Idea 3. Score each website equal to the weighted sum, over all other webpages linking to it, of the reciprocal of the number of outgoing links from that other page, weighted by the score of that other page.

\[ \text{score}_3(v) = \sum_{u: (u,v) \in E} \frac{\text{score}_3(u)}{d^+(u)}. \]

\( \text{score}_3(v) \) seems to attain some kind 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. The total authority distributed by a webpage \( u \) is exactly equal to \( u \)'s own authority, \( \text{score}_3(u) \). But while this recursive relationship is appealing, there is no reason, \textit{a priori}, why such scores should exist.

Today's discussion is about how \( \{\text{score}_3(v), v \in V\} \) \textit{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.\footnote{We note that the mathematics we discuss existed long before search engines and similar ideas had been applied elsewhere.}

The goal of this discussion is to prove the following theorem (in more general terms). Recall that \( \Delta^V = \{x \in \mathbb{R}^V_\geq 0 : \sum_{v \in V} x_v = 1\} \) denotes the set of probability vectors over \( V \).
Theorem 14.1. There exists a vector \( x \in \Delta^V \) such that

\[
x_v = \sum_{u:(u,v) \in E} \frac{x_u}{d^+(u)}
\]

for all \( v \in V \). If \( G \) is strongly connected, then this vector is unique and strictly positive.

It is already surprising that such an \( x \) always exists. In addition to this, we are promised that \( x \) is a probability vector \( V \). Coincidence?

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 choosing links. Note that if the links tend to point to useful websites, then over time your random walk should be better than uniformly random at stumbling upon good web sites. Now, depending on where you start, after some number of \( k \) steps, there are different probabilities of where you would end up. We might hope this reaches some kind of equilibrium, where the distribution is the same or close to the same between to \( k \)th and \((k+1)\)th step for sufficiently large \( k \). Let us thus define the notion of a stationary distribution.

Definition 14.2. 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 14.1, 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 vector \( u \), we choose an outgoing edge \((u,v)\) with probability \( 1/d^+(u) \). In particular, if we take a step 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 14.1, this sum equals \( x_v \). That is, \( x \) is the stationary distribution of a random walk on \( G \). We can restate theorem 14.1 as follows.
Theorem 14.1, restated. Every random walk on a directed graph $G$ has a stationary distribution. If $G$ is strongly connected, then this distribution is unique.

14.1.1 PageRank

The actual PageRank algorithm proposed in pagerank is slightly different. We augment the random walk through experiment described above with a small probability $\alpha$ 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 14.3.** 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 $v \in V$,

$$x_v = (1 - \alpha) \sum_{u, (u, v) \in E} \frac{x_u}{d^+(u)} + \frac{\alpha n}{n}.$$

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 14.1 is unique.) Second, it can be calculated more directly. We will come back to this point at the end of our discussion in section 14.6.
14.2 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 $v$th coordinate. We define $A$ by setting $(Ae_v) \in \{0, 1\}^V$ to indicate the outgoing neighbors 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. 14.3. 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

$$\langle e_w, Ae_v \rangle = (Ae_v)_w = \begin{cases} \text{weight of the edge } (v, w) & \text{if } (v, w) \in E \\ 0 & \text{otherwise.} \end{cases}$$
Let us rehash our discussion on scoring websites in terms of the linear map. We have
\[ \text{score}_1(v) = \langle e_v, A1 \rangle = \sum_w (Ae_v)_w. \]

Let
\[ d^+(u) = \langle 1, Ae_u \rangle. \]

Let \( D = \text{diag}(d^+) \in \mathbb{R}^{V \times V} \) 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 \( \text{score}_2(v) \) as
\[ \text{score}_2(v) = \sum_{u:(u,v) \in E} \frac{1}{d^+(u)} = (AD^{-1}1)_v = \langle e_v, AD^{-1}1 \rangle. \]

As for the stationary distribution (\( \text{score}_3 \)), we see that the stationary distribution \( x \) must satisfy the equation
\[ x = Rx \]
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 \( \Delta^n \) to \( \Delta^n \) is called a \textit{stochastic} linear function.

### 14.3 Eigenvectors

Let \( A : \mathbb{R}^n \to \mathbb{R}^n \) be a linear map. An \textit{eigenvector} of \( A \) is a nonzero vector \( x \neq 0 \) such that
\[ Ax = \lambda x \]
for some scalar value \( \lambda \in \mathbb{C} \). The scalar \( \lambda \) is called an \textit{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 \( \lambda \) gives a second vector (besides \( 0 \)) such that \( (A - \lambda I)x = 0 \).

**Lemma 14.4.** The set of eigenvectors corresponding to an eigenvalue \( \lambda \) form a vector space.

The dimension of the subspace corresponding to an eigenvalue \( \lambda \) is called the \textit{multiplicity} of the eigenvalue. For any eigenvalue \( \lambda \) of \( A : \mathbb{R}^n \to \mathbb{R}^n \), the multiplicity of \( \lambda \) is equal to \( n - \text{rank}(A - \lambda I) \).
Existence of eigenvectors. \textit{A priori}, it is not clear why every matrix should have an eigenvector.

**Lemma 14.5.** 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 $v, Av, \ldots, A^n v$ 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 $r_1, \ldots, r_n \in \mathbb{C}$ are complex roots of $p(x)$. Then

$$p(A)v = (A - r_1 I)(A - r_2 I) \cdots (A - r_k I)v = 0$$

implies that some $A - r_i I$ maps a nonzero vector to 0. The corresponding root $r_i$ is an eigenvalue. \hfill \Box

**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 $\lambda$ for which $A - \lambda I$ is not full rank.

**Lemma 14.6.** 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

$$\langle x, (A^{-1})^T A^T y \rangle = \langle A^{-1} x, A y \rangle = \langle A A^{-1} x, y \rangle = \langle x, y \rangle.$$ 

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 $(A^T)^T = A$. \hfill \Box

**Lemma 14.7.** 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 $\lambda$ 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 $\lambda$ is an eigenvalue of $A$. The multiplicities are equal because the

$$\text{rank}(A - \lambda I) = \text{rank}((A - \lambda I)^T) = \text{rank}(A - \lambda I).$$

\[ \square \]

Lemma 14.8. 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 $\langle x, y \rangle = 0$.

Proof. Let $x$ have corresponding eigenvalue $\lambda$ and let $y$ have corresponding eigenvalue $\mu$. We have

$$\lambda \langle x, y \rangle = \langle Ax, y \rangle = \langle x, A^T y \rangle = \mu \langle x, y \rangle.$$

If $\lambda \neq \mu$, then $\langle x, y \rangle = 0$. \[ \square \]

Let us now restate theorem 14.1 in our new language of eigenvectors and eigenvalues.

**Theorem 14.1, 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.

14.4 The Perron-Frobenius theorem

**Definition 14.9.** 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 $x, y \neq 0$.

The following theorem is called the Perron-Frobenius theorem.

**Theorem 14.10.** Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map such that $Ax > 0$ (coordinatewise) for all $x \in \mathbb{R}^n_{\geq 0}$ with $x \neq 0$. Then $A$ has an eigenvalue $\lambda_1$ with eigenvector $x_1$ with the following properties.

1. $\lambda_1 > 0$ and $x_1 > 0$.

2. $x_1$ is the unique (generalized) eigenvector of $\lambda_1$.

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 \( \mu \) has \( |\mu| < x_1. \)

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 supremum of \( L \) is the desired value \( \lambda_1 \). To this end, we first make the following claims about \( L \).

1. \( L \) is nonempty, and contains a positive number.
2. \( L \) is bounded.
3. \( L \) is closed.

For claim 1, let \( x \in \Delta^n \) with \( x > 0 \). The \( Ax \) will have strictly positive coordinates because the graph is strongly connected, and so there is some \( \lambda > 0 \) such that \( Ax > \lambda x \).

For claim 2, we observe that for \( \lambda \in L \), with \( Ax \geq \lambda x \) with \( x \in \Delta^n \), we have

\[ \langle 1, A1 \rangle \geq \langle 1, Ax \rangle \geq \langle 1, \lambda x \rangle = \lambda. \]

(a) is because \( A \) is monotonic and \( 1 \geq x \). (b) is by choice of \( x \) and \( \lambda \). (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 \( \lambda \in L \). For each \( i \), let \( x_i \in \Delta^n \) with \( Ax_i \geq \lambda_i x_i \). Since \( \Delta^n \) converges, the \( x_i \)'s converge to some \( \bar{x} \in \Delta^n \). We have

\[ A\bar{x} = A \left( \lim_{n \to \infty} x_i \right) = \lim_{n \to \infty} Ax_i \geq \lim_{n \to \infty} \lambda_i x_i = \lambda \bar{x}. \]

(d) invokes continuity of \( A \) to pass through the limit.

We have now show 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, \( \lambda_1 \), 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 \( Ax_1 = \lambda_1 x_1 \).

Let \( y \geq 0 \) be such that \( Ax_1 = \lambda_1 (x_1 + y) \). Then \( Ax_1 = \lambda_1 x_1 \iff y = 0 \). Suppose by contradiction that \( y \neq 0 \). Choose \( \epsilon \in (0, 1) \) sufficiently small that

\[ \frac{\epsilon}{1 - \epsilon} \leq \langle 1, y \rangle. \]

\footnote{We will only prove that \( |\mu| < x_1 \).}
Observe that

\[(1 - \epsilon)x_1 + \frac{\epsilon}{\langle 1, y \rangle} \in \Delta^n.\]

Moreover, we have

\[A \left( (1 - \epsilon)x_1 + \frac{\epsilon}{\langle 1, y \rangle} y \right) \geq (1 - \epsilon)Ax_1 \overset{(g)}{=} (1 - \epsilon)\lambda(x_1 + y) \geq \lambda \left( (1 - \epsilon)x_1 + \frac{\epsilon}{\langle 1, y \rangle} y \right).\]

Here (e) is because $Ax > 0$ for all $x \geq 0$ with $x \neq 0$. (f) is by choice of $y$. (g) is by choice of $\epsilon$. The strict inequality obtained above implies that there is a larger value than $\lambda$ in $L$, a contradiction. Thus we must have $Ax_1 = A\lambda_1$ after all.

Note also that $x_1$ is strictly positive, as $x_1 = \lambda^{-1}Ax_1 > 0$. 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 $\lambda_1$ (up to scaling). Indeed, suppose $Ay = \lambda_1 y$ for vector $y$. As mentioned above, we must have $y > 0$. If $y$ is not proportional to $x$, then let $z = x - \alpha y$ where $\alpha > 0$ is such that $z \geq 0$, $z \neq 0$, 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 $\lambda_1$, 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 $\lambda_1$, so

\[Ay = \lambda_1 y + cx_1\]  \hfill (14.1)

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 (1) that there is a value larger than $\lambda_1$ in $L$, a contradiction. Thus $x_1$ is the unique generalized eigenvector for $\lambda_1$.

Take any other eigenvalue $\mu$ 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 $\theta$. Thus $y \in \text{span}(x_1)$ and $\mu = x_1$. 

207
For the last claim, observe that $A^T$ also satisfies the hypothesis. Indeed, if $x, y \geq 0$ and neither $x$ nor $y$ equals zero, then
\[
\langle A^T x, y \rangle = \langle x, Ay \rangle > 0.
\]
Thus $A^T$ has (the same) dominant eigenvalue $\lambda_1$, 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.

The above proof is essentially due to Bohnenblust; see [lax; bellman].

### 14.5 Perron-Frobenius for strongly connected random walks

We now extend theorem 14.10 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 14.11.** 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 $x > 0$.
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 $\mu$ 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 $i \in \{0, \ldots, n-1\}$ uniformly at random.
14. Random walks

14.6. Computing PageRank

2. Take \( i \) steps of the random walk \( A \).

Observe that any eigenvalue \( \mu \) 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 14.10.

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 \( \lambda_1 \), with a corresponding eigenvector that is also the only eigenvector with no negative coordinates. We also know that 1 is an eigenvalue of \( B^T \) with eigenvector \( 1 \). So \( \lambda_1 = 1 \). Theorem 14.10 then implies that \( x \) is the unique eigenvector for eigenvalue 1, and all the coordinates of \( x \) are strictly positive.

Let \( \mu \) 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

\[
\mu_\epsilon = \mu + \frac{\epsilon}{n} \sum_{i=1}^{n-1} \mu^i.
\]

\( \mu_\epsilon \) is an eigenvalue of \( A_\epsilon \) for all \( \epsilon \geq 0 \). Note that for all \( \epsilon > 0 \), theorem 14.10 applies to \( A_\epsilon \) and in particular \( A_\epsilon \) 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. \( \square \)

14.6 Computing PageRank

The PageRank vector \( x \) satisfies the equation

\[
x = (1 - \epsilon)Rx + \frac{\epsilon}{n} 1,
\]

where \( R = AD^{-1} \) models the random walk on the directed graph, and where \( 1 \in \mathbb{R}^V \) is the all-1’s vector. We can rewrite this as

\[
\left(\left(\frac{1}{1 - \epsilon}I - R\right)x = \frac{\epsilon}{(1 - \epsilon)n} 1.
\]

209
Recall that the maximum eigenvalue of $R$ is 1; in particular, $1/(1 - \epsilon)$ is not an eigenvalue. Thus $(1/(1 - \epsilon)I - R)$ is invertible. We have

$$x = \left( \frac{\epsilon}{(1 - \epsilon)n} \right) \left( \frac{1}{1 - \epsilon} I - R \right)^{-1} = \frac{\epsilon}{n} (I - (1 - \epsilon)R)^{-1} \mathbb{I}.$$  

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

$$x = \frac{\epsilon}{(1 - \epsilon)n} \lim_{k \to \infty} \sum_{i=0}^{k} ((1 - \epsilon)R)^{i} \mathbb{I}.$$  

The series on the RHS converges quickly for moderate $\epsilon$, so in practice one only has to compute a few terms in the sum.

### 14.7 Exercises

**Exercise 14.1.** Let $G = (V, E)$ be a directed graph, not necessarily strongly connected. Recall that the strongly connected component for a directed acyclic graph. A *sink component* is a strongly connected component with no outgoing 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 14.2.** Let $G = (V, E)$ be a simple\(^3\), unweighted, directed, and strongly connected graph. We proved in theorem 14.1 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 15

Connectivity and Electricity

15.1 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. The common feature to these algorithms is that they mark the vertices that they visit, so they don’t revisit them. 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. The search algorithms like BFS or DFS mark the vertices. This leads to $O(n)$ total space. 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 [Savitch1970]. 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 15.1.** 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 15.2.** 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 18.

### 15.2 Electrical networks

Our analysis will be based on a (perhaps surprising) connection between random walks an 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 the 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, than just the sum.

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 \((s,t)\)-flow \( f \) minimizing the 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 following computer. (The following 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 $\langle 1, d \rangle = 0$). In the above algebraic notation, the electrical flow is obtained as the solution to the following optimization problem:

$$\minimize \langle f, Rf \rangle = \sum_{e \in E} r_e f_e^2 \text{ over } f \in \mathbb{R}^E \text{ s.t. } Bf = d. \quad (15.1)$$

For a fixed electrical network, the quantity above is a function of $d$. In general, for $d \in \mathbb{R}^V$ with $\langle 1, d \rangle = 0$, 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 (15.1).

2. The second part is about computing electrical flows. Remarkably, one can compute electrical flows in nearly linear time!

### 15.3 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.

#### 15.3.1 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 15.3. 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 \(\ker(A) = \{x : Ax = 0\}\) denote the kernel of \(A\); i.e., the set of vectors that map to 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 \(\text{im}(A)\), is the set

\[\text{im}(A) \overset{\text{def}}{=} \{Ax : x \in X\},\]

and that the coimage of \(A\), denoted \(\text{coim}(A)\), is the subspace of \(X\) orthogonal to \(\ker(A)\):

\[\text{coim}(A) \overset{\text{def}}{=} X/\ker(A) = \{x \in X : \langle x, y \rangle = 0 \text{ for all } y \in \ker(A)\}.\]

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 \(\text{coim}(A)\) and \(\text{im}(A)\). The one-to-one mapping \(A^T : \text{im}(A) \to \text{coim}(A)\) implies that there exists \(y \in Y\) such that \(A^Ty = \varphi'(x).\)
15.3.2 Ohm’s Law

If we apply theorem 15.3 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 15.4.** 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 $f = R^{-1}B^Tp$.

**Proof.** Suppose $f$ is the electrical flow. Observe that the gradient of the objective function is $Rf$. By theorem 15.3, there exists $q \in \mathbb{R}^V$ such that $2Rf = Bq$; hence $p = q/2$ is the desired set of electrical potentials.

Conversely, suppose $f = R^{-1}B^Tp$ for some $p \in \mathbb{R}^V$. Let $g$ be any other flow with $Bg = d$. Recall that for any convex function $\varphi$, 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

$$\langle g, Rg \rangle - \langle f, Rf \rangle \overset{(a)}{\geq} 2\langle Rf, g - f \rangle \overset{(b)}{=} 2\langle B^Tp, g - f \rangle = 2\langle p, B(g - f) \rangle = 2\langle p, d - d \rangle = 0.$$  

Here (a) applies convexity of $\varphi(x)$, and (b) substitutes $r = R^{-1}Bp$. \hfill \Box

15.4 Effective resistance and the Laplacian

Recall that the Laplacian of a graph $G$ with edge weights $w(e)$ is the symmetric matrix $L : \mathbb{R}^V \rightarrow \mathbb{R}^V$ defined by

$$\langle x, Lx \rangle = \sum_{e \in \{u,v\} \in E} w(e)(x_u - x_v)^2.$$  

**Lemma 15.5.** Let $w(e) = 1/r_e$ for all $e$. Then $L = BR^{-1}B^T$.

To prove lemma 15.5, since both matrices are symmetric, it suffices to show that $\langle x, Lx \rangle = \langle x, BR^{-1}B^Tx \rangle$ for all $x$. We leave this calculation to the reader as exercise 15.3.

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., $1$.  

216
**Theorem 15.6.** 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

\[
\langle f, Rf \rangle \overset{(a)}{=} \langle R^{-1}B^T p, RR^{-1}B^T p \rangle = \langle p, Lp \rangle = \langle Lp, L^{-1}Lp \rangle \overset{(b)}{=} \langle d, L^{-1}d \rangle
\]

where (a) is by Ohm’s Law.

\( \square \)

### 15.5 Effective conductance

Consider the following optimization problem.

\[
\text{minimize } \langle p, Lp \rangle \text{ over } \langle p, d \rangle = 1. \tag{15.2}
\]

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 (15.2) is sometimes called the effective conductance.

The first order conditions tell us that the optimum solution \( p^* \) satisfies \( 2Lp^* = \lambda d \), hence \( p^* = (\lambda/2)L^{-1}d \), for some scalar \( \lambda \). To identify \( \lambda \), we plug into \( \langle p^*, d \rangle = 1 \): We have

\[
1 = \langle p^*, d \rangle = \frac{\lambda}{2} \langle d, L^{-1}d \rangle,
\]

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 (15.2), we have

\[
\langle p^*, Lp^* \rangle = \langle Lp^*, L^{-1}Lp^* \rangle = \frac{\lambda^2}{4} \langle d, L^{-1}d \rangle = \frac{1}{\langle d, L^{-1}d \rangle}.
\]

The following theorem summarizes our developments.
Theorem 15.7. (15.2) 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 15.8. 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 15.9. 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 1).

15.6 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 15.1. 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 15.10.** 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 15.11.** Let \( G = (V,E) \) be an undirected graph with \( m \) edges and \( n \) vertices. Let \( s,t \in V \).

\[
H(s,t) + H(t,s) = 2m(\text{effective resistance from } s \text{ to } t).
\]

**Proof.** Let \( p = H(\cdot, s) \) and \( q = H(\cdot, t) \) be the electrical potentials encoding the hitting times to \( s \) and to \( t \), respectively. 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

$$H(s, t) + H(t, s) = \langle r, d \rangle = \frac{1}{2m} \langle r, BB^T r \rangle = \frac{1}{2m} \langle B^T r, B^T r \rangle = 2m \langle (1/2m) Br, (1/2m) Br \rangle,$$

as desired. 

\[ \square \]

**Theorem 15.12.** 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 15.11 as an upper bound on the effective resistance gives the bound we seek. \[ \square \]

**Theorem 15.2.** 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 $C(s) \leq m(n - 1)$. 

220
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 $e = (u, v) \in T$ is $H(u, v)$. The total time over the entire spanning tree is

$$
\sum_{e=\{u,v\}\in T} H(u,v) + H(v,u) \leq \sum_{e=\{u,v\}\in T} 2m \leq 3m(n-1).
$$

15.7 Exercises

Exercise 15.1. 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 15.2. 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 15.3. Complete the proof of lemma 15.5.
Chapter 16

Undirected Graphs and the Spectral Theorem

16.1 The Laplacian of a graph

In our discussion on electrical networks, we briefly came across the Laplacian form of an undirected graph. Let us reintroduce the Laplacian $L : \mathbb{R}^V \to \mathbb{R}^V$ of a weighted undirected graph $G = (V, E)$, this time from a different perspective.

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 corresponding weighted sum of Laplacians of the edges,

$$L = \sum_e w(e) L_e.$$

Given an input vector $x \in \mathbb{R}^V$, we have

$$\langle x, Lx \rangle = \sum_{e \in E} w(e) \langle x, L_e x \rangle = \sum_{e = (u, v) \in E} w(e) (x_u - x_v)^2.$$

That is, the $L$ induces a simple sum of squared differences on $x$, based on the edges of the graph.

\(^1\)We are avoiding the conventional notation $e_u$ for the standard basis vectors because $e$ is so frequently used for edges.
In fact, it induces a very familiar sum of squares. Recall that the electrical flow problem is to minimize $\langle f, Rf \rangle$ over $f \in \mathbb{R}^E$ s.t. $Bf = d$. Here $R = \text{diag}(r)$ is the diagonal map of resistances $r \in \mathbb{R}_{>0}^E$. $d \in \mathbb{R}^V$ 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 $f = B^T p$ for a set of vertex potentials $p$. Then we have

$$\langle f, Rf \rangle = \langle B^T p, R B^T p \rangle = \sum_{e=(u,v) \in E} r_e (p_u - p_v)^2.$$ 

That is, we are choosing $p$ as to minimize the Laplacian of the graph with edge weights corresponding to the resistances.

Recall that a linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is symmetric if $A = A^T$. It is easy to see that the Laplacian $L$ is symmetric: each $L_e$ is symmetric since in general $(a \otimes b)^T = (b \otimes a)$, 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 16.1.** A linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is a positive semi-definite linear operator if

1. $A$ is symmetric.
2. $\langle x, Ax \rangle \geq 0$ for all $x \in \mathbb{R}^n$.

$A$ is (strictly) positive definite if in addition to being positive semi-definite,

3. $A$ is invertible.

The Laplacian $L$ is not invertible: $L 1 = 0$. If $G$ is connected, and we restrict to the $n-1$ space $\mathbb{R}^V / 1$, then $L$ is invertible and (strictly) positive definite (see ??).

### 16.2 The Spectral Theorem for Symmetric Maps

We note about that the Laplacian $L : \mathbb{R}^V \to \mathbb{R}^V$ of an undirected graph is a symmetric and positive semi-definite map. We have seen before in our discussion of random walks that the eigenvectors can be very insightful in understanding the behavior of
a linear map. We will see here that the eigenvectors of symmetric linear maps are particularly well behaved.

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 $\lambda \in \mathbb{C}$ of $Ax = \lambda x$. Recall the following facts proven in our earlier discussions, which apply generally to all linear maps.

**Fact 16.2.** Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. Then $A$ has an eigenvalue $\lambda \in \mathbb{C}$ 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.

**Lemma 16.3.** 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 $Lx = \lambda x$ for $\lambda = \langle x, Lx \rangle$.

**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 $Lx \in \text{span}(x)$; i.e., $Lx = \lambda x$ for some $\lambda \in \mathbb{R}$. Upon inspection, $\lambda = \lambda \langle x, x \rangle = \langle x, Lx \rangle$, as claimed.

Let $u \in X$ with $\|u\| = 1$ and $\langle u, x \rangle = 0$. Define

$$f(\epsilon) = \langle \frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}}, L\left(\frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}}\right) \rangle = \frac{\langle x + \epsilon u, L(x + \epsilon u) \rangle}{1 + \epsilon^2}.$$

$f(\epsilon)$ 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 $\frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}}$ 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 17.2.)

**Remark 16.4.** 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 above lemma implies that beyond simply having an eigenvalue, a symmetric map has a real-valued eigenvalue without extending the field to include complex values. This simple fact leads to the following sweeping theorem about symmetric matrices.

**Theorem 16.5.** 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 $u_1, \ldots, u_n$ of $X$ and $n$ scalar values $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ such that

$$A = \lambda_1(u_1 \otimes u_1) + \lambda_2(u_2 \otimes u_2) + \cdots + \lambda_n(u_n \otimes u_n).$$
Proof. If \( n = 0 \), then the claim is tautological, as \( X \) is the trivial vector space \( \{0\} \) and \( A \) can be expressed as an empty sum. Suppose \( n \geq 1 \). By lemma 16.3, \( A \) has a real-valued eigenvalue \( \lambda \) with a corresponding eigenvector \( u \in X \). By scaling \( u \), we may assume \( \|u\| = 1 \). Consider the map \( B = A - \lambda(u \otimes u) \). \( B \) is also symmetric, and maps the space \( \text{span}(x) = \{\alpha x : \alpha \in \mathbb{R}\} \) to \( 0 \). 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
\[
\langle x, By \rangle = \langle Bx, y \rangle = \langle 0, y \rangle = 0,
\]
so \( By \in Y \).

Thus \( B \) restricts to a linear and symmetric operator on \( Y \). By induction on \( n \), there is an orthonormal basis \( u_1, \ldots, u_{n-1} \) of \( Y \) and scalar values \( \lambda_1, \ldots, \lambda_{n-1} \in \mathbb{R} \) such that
\[
B = \lambda_1(u_1 \otimes u_1) + \cdots + \lambda_{n-1}(u_{n-1} \otimes u_{n-1}).
\]
Let \( \lambda_n = \lambda \) and \( u_n = u \). Observe that \( u_1, \ldots, u_n \) is an orthonormal basis of \( X \). We have
\[
\lambda_1(u_1 \otimes u_1) + \cdots + \lambda_n(u_n \otimes u_n) = B + \lambda_n(u_n \otimes u_n) = A,
\]
as desired.

Theorem 16.5 makes the structure of any symmetric map \( A : \mathbb{R}^n \to \mathbb{R}^n \) extremely transparent. By theorem 16.5, let \( u_1, \ldots, u_n \in \mathbb{R}^n \) and \( \lambda_1, \ldots, \lambda_n \in \mathbb{R} \) be such that
\[
A = \lambda_1(u_1 \otimes u_1) + \cdots + \lambda_n(u_n \otimes u_n).
\]
It will be convenient to assume that that \( \lambda_i \)'s are in nonincreasing order: \( \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \).

For any input vector \( x \in \mathbb{R}^n \), we can write \( x \) uniquely in the form
\[
x = \alpha_1 u_1 + \cdots + \alpha_n u_n;
\]
namely, with \( \alpha_i = \langle x, u_i \rangle \). Then we have
\[
Ax = \lambda_1(u_1 \otimes u_1)x + \cdots + \lambda_n(u_n \otimes u_n)x = \lambda_1 \alpha_1 u_1 + \cdots + \lambda_n \alpha_n u_n.
\]
That is, in the basis \( \{u_1, \ldots, u_n\} \), \( A \) simply rescales the \( i \)th coordinate by a factor of \( \lambda_i \). That is to say:
Every symmetric matrix is a diagonal matrix up to a rotation (i.e., change in basis).

We can see from the construction in the proof that the $\lambda_i$'s are the eigenvalues of $A$ and the $u_i$'s are eigenvectors. But this fact is even more obvious in hindsight given the representation

For each $i$, we have

$$Au_i = \lambda_i (u_i \otimes u_i) u_i = \lambda_i u_i,$$

by orthonormality of the $u_i$'s.

**Theorem 16.6.** Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a symmetric linear operator. Let $\lambda_1, \ldots, \lambda_n$ be the $n$ eigenvalues of $A$ (with multiplicity) in decreasing order. Then

$$\lambda_k = \min_{S : \dim(S) = k-1} \max_{x \in X/S} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.$$

### 16.3 The Laplacian and cuts

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 $G$. A helpful way to interpret to Laplacian is to see that it encodes all the cuts of $G$. Let $S \subset V$ be a set of vertices, and let $\mathbbm{1}_S \in \{0, 1\}^V$ be the indicator vector for $S$. Then we can extract the weight of the cut $\delta(S)$ from the Laplacian as

$$\langle \mathbbm{1}_S, L \mathbbm{1}_S \rangle = \sum_{e \in \delta(S)} w(e).$$

Recall that the (uniform) sparsest cut of an $G$ is the set $S$ minimizing the ratio

$$\frac{\sum_{e \in \delta(S)} w(e)}{|S|(n - |S|)}.$$

The minimum such quantity over a graph $G$ is called the sparsity of $G$ and denoted

$$\Psi(G) = \min_{S \subset V} \frac{\sum_{e \in \delta(S)} w(e)}{|S|(n - |S|)}.$$

Let us now relate cuts to some of the eigenvectors of $L$. Observe that because $L$ is positive semi-definite, all its eigenvalues are nonnegative. Moreover, we know that $\mathbbm{1}$
is an eigenvector with eigenvalue of 0—this gives us our smallest eigenvector. The eigenvector corresponding to the second smallest eigenvalue, denoted $\lambda_2$, is given by

$$
\lambda_2 = \min_{x: \langle x, x \rangle = 0} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.
$$

Now, consider any cut $\mathbf{1}_S$, and let $x$ be the orthogonal projection from $\mathbf{1}$; namely,

$$
x = \mathbf{1}_S - \alpha \mathbf{1} \text{ where } \alpha = \langle \mathbf{1}_S, \mathbf{1} \rangle / \langle \mathbf{1}, \mathbf{1} \rangle = \frac{|S|}{n},
$$

Observe that

$$
\langle x, x \rangle = \langle x, \mathbf{1}_S \rangle = (1 - \alpha) |S| = (n - |S|) |S| / n.
$$

Thus

$$
\lambda_2 \leq \frac{n \sum_{e \in \delta(S)} w(e)}{|S| (n - |S|)}.
$$

Taking the minimum over all sets $S$, we obtain the following.

**Theorem 16.7.** 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 $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of $G$. Let $\lambda_2$ be the second smallest eigenvalue of $L$ and let $\Psi(G)$ be the sparsity of $G$. Then

$$
\lambda_2 \leq n \Psi(G).
$$

We will come back to this discussion later.

### 16.4 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 $D = \text{diag}(A\mathbf{1})$ is the diagonal map of weighted vertex degrees. Recall that $R$ was the beneficiary of 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 value 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 16.8. 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 $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map of $G$. Then $R$ has stationary distribution proportional to the weighted degrees of its vertices.

Proof. We have


While $R$ is not a symmetric map\footnote{unless $G$ is regular; see exercise 16.3}, we can extend the spectral theorem for symmetric maps to $R$ by way of similarity.

Definition 16.9. Two linear maps $A, B : X \to X$ are similar if $A = C^{-1}BC$ for an invertible map $C : X \to X$.

Lemma 16.10. Let $A$ and $B$ be similar. Then their kernels are isomorphic. In particular, if $A = C^{-1}BC$, then $C$ restricts is an isomorphism between $\ker(A)$ and $\ker(B)$.

Lemma 16.11. 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 $\lambda$, $A - \lambda I$ 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 $1 = \lambda_1, \ldots, \lambda_n \geq -1$ list the eigenvalues of $Q$ in decreasing order. Let $u_1, \ldots, u_n$ be an orthonormal basis of $\mathbb{R}^V$ such that

$$Q = u_1 \otimes u_1 + \lambda_2(u_2 \otimes u_2) + \cdots + \lambda_n(u_n \otimes u_n).$$
16. Undirected Graphs and the Spectral Theorem

16.4. Random walks in undirected graphs

(Here we substituted $\lambda_1 = 1$). We also know, from ??, that $D^{1/2}u_1$ must correspond to the uniform distribution, which is proportional to $d$. Since $u_1$ is has unit length, we have

$$u_1 = \frac{D^{-1/2}(d)}{\|D^{-1/2}d\|} = \frac{1}{\sqrt{2W}} \sqrt{d},$$

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

$$Q = \frac{1}{2W} \left( \sqrt{d} \otimes \sqrt{d} \right) + \lambda_2 (u_2 \otimes u_2) + \cdots + \lambda_n (u_n \otimes u_n).$$

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

$$R^k x \overset{(a)}{=} D^{1/2} Q^k D^{-1/2} x$$

$$= D^{1/2} \left( \frac{1}{2W} \left( \sqrt{d} \otimes \sqrt{d} \right) + \lambda_2 (u_2 \otimes u_2) + \cdots + \lambda_n (u_n \otimes u_n) \right)^k D^{-1/2} x$$

$$\overset{(b)}{=} D^{1/2} \left( \frac{1}{2W} \left( \sqrt{d} \otimes \sqrt{d} \right) + \lambda_2^k (u_2 \otimes u_2) + \cdots + \lambda_n^k (u_n \otimes u_n) \right) D^{-1/2} x$$

$$\overset{(c)}{=} \frac{d}{2W} + \left( \lambda_2^k (u_2 \otimes u_2) + \cdots + \lambda_n^k (u_n \otimes u_n) \right) D^{-1/2} x$$

(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 orthonormal\(^3\) – here we see some of the power of the spectral theorem! (c) is because

$$D^{1/2} \left( \sqrt{d} \otimes \sqrt{d} \right) D^{-1/2} x = (d \otimes 1)x = \langle 1, x \rangle d = d.$$

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

$$D^{1/2} \left( \lambda_2^k (u_2 \otimes u_2) + \cdots + \lambda_n^k (u_n \otimes u_n) \right) D^{-1/2} x.$$
Let $S = \lambda_2^k (u_2 \otimes u_2) + \cdots + \lambda_n^k (u_n \otimes u_n)$; $S$ is symmetric, with eigenvalues $0, \lambda_2^k, \ldots, \lambda_n^k$. Let $\Delta_{\max}$ be the maximum degree in $G$ and let $\Delta_{\min}$.

Let $\Delta_{\max}$ be the maximum degree in $G$ and let $\Delta_{\min}$.

$$\| R^k x - \frac{1}{2W} d \|^2 = \| D^{1/2} S D^{-1/2} x \|^2 = \langle S D^{-1/2} x, D (S D^{-1/2} x) \rangle \leq \Delta_{\max} \| S D^{-1/2} x \|^2 = \Delta_{\max} \langle D^{-1/2} x, S^2 (D^{-1/2} x) \rangle \leq \max \{ \lambda_2^{2k}, \lambda_n^{2k} \} \Delta_{\max} \langle x, D^{-1/2} x \rangle \leq \Delta_{\max} \langle x, D^{-1} x \rangle \leq \max \{ \lambda_2^{2k}, \lambda_n^{2k} \} \Delta_{\max} \Delta_{\min}.$$

(d) and (e) both follow from the fact that for a symmetric map $A$ with maximum eigenvalue $\lambda$, we have $\langle x, Ax \rangle \leq \lambda \| x \|^2$ for all $x$ (lemma 16.3). $D$ has maximum eigenvalue $\Delta_{\max}$, $S^3$ has maximum eigenvalue $\max \{ \lambda_2^{2k}, \lambda_n^{2k} \}$, and $D^{-1}$ has maximum eigenvalue $1/\Delta_{\min}$. Recall that $\lambda_2, \lambda_n \in [-1, 1]$. If $\lambda_2$ and $\lambda_n$ are both bounded away from both 1 and $-1$, then $\max \{ \lambda_2^{2k}, \lambda_n^{2k} \} = \max \{ \lambda_2, |\lambda_n| \}^{2k} \to 0$ as $k \to \infty$. To this end, the spectral gap of a random walk $R$ is defined as the difference

$$\gamma = 1 - \max \{ \lambda_2, |\lambda_n| \},$$

where $\lambda_2$ is the second largest eigenvalue and $\lambda_n$ is the smallest eigenvalue. We have given the following bound on the convergence rate as a function of the spectral gap.

**Theorem 16.12.** 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 $R : \mathbb{R}^V \rightarrow \mathbb{R}^V$ be the random walk map of $G$ and let $\gamma$ 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

$$\| R^k x - s \| \leq (1 - \gamma)^k \sqrt{\frac{\Delta_{\max}}{\Delta_{\min}}}.$$

16.5 Exercises

**Exercise 16.1.** 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 $L : \mathbb{R}^V \rightarrow \mathbb{R}^V$ be the Laplacian of $G$. 230
Prove that $G$ is connected iff for any $x \not\in \text{span}(1)$, we have $\langle x, Lx \rangle > 0$.

**Exercise 16.2.** Finish the proof of lemma 16.3, 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 16.3.** 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 $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map. Prove that $R$ is symmetric iff $G$ is regular.

**Exercise 16.4.** Suppose your goal was to converge to the uniform distribution via a random walk for a single vertex as fast as possible. Show that one can choose a vertex $v$ such that, starting from an initial distribution of $x = 1_v$, the $\ell_2$-distance from the stationary distribution after $k$ steps is at most $(1 - \gamma)^k$.

**Exercise 16.5.** 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 $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of $G$. Suppose the spectral gap $\gamma$ 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 $G$.

---

4A graph is regular if every vertex has the same weighted degree
Chapter 17

Conductance

17.1 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.

17.2 The Laplacian of a graph

As mentioned above, we are interested in a certain matrix representation of a graph called the Laplacian. We first show how to model a single edge by a 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, representing pairwise relations between vertices, are modeled as matrices in $\mathbb{R}^{V \times V}$.  

232
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 corresponding weighted sum of Laplacians of the edges,

$$L = \sum_e w(e) L_e.$$

Given an input vector $x \in \mathbb{R}^V$, we have

$$\langle x, Lx \rangle = \sum_{e \in E} w(e) \langle x, L_e x \rangle = \sum_{e = (u, v) \in E} w(e) (x_u - x_v)^2.$$

That is, $L$ induces a simple sum of squared differences on $x$, based on the edges of the graph. We will see that $\langle x, Lx \rangle$ encodes a lot of information about $G$. One example is that $L$ encodes the weight of all the cuts. Given a set $S \subset V$, if we let $1_S$ denote the indicator vector of $S$, we have

$$\langle 1_S, L 1_S \rangle = \sum_{e \in \delta(S)} w(e).$$

Another interpretation of $\langle x, Lx \rangle$ that we discuss in greater detail later is based on electrical networks. $\langle x, Lx \rangle$ is the electrical energy of an electrical network where the edge weights give resistances and $x$ is a set of vertex potentials.

Recall that a linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is symmetric if $A = A^T$. It is easy to see that the Laplacian $L$ is symmetric: each $L_e$ is symmetric since in general $(a \otimes b)^T = (b \otimes a)$, 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.

---

\(^1\)We are avoiding the conventional notation $e_u$ for the standard basis vectors because $e$ is so frequently used for edges.
**Definition 17.1.** A linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is a positive semi-definite linear operator if

1. $A$ is symmetric.
2. $\langle x, Ax \rangle \geq 0$ for all $x \in \mathbb{R}^n$.

A is (strictly) positive definite if in addition to being positive semi-definite,

3. $A$ is invertible.

The Laplacian $L$ is not invertible: $L1 = 0$. If $G$ is connected, and we restrict to the $n - 1$ space $\mathbb{R}^V / 1$, then $L$ is invertible and (strictly) positive definite (see ??).

### 17.3 Sparse cuts

Recall that the sparsity of a cut $\delta(S)$ (where $S \subset V$), which we denote $\Phi(S)$, was the ratio

$$\Phi(S) \triangleq \frac{\bar{w}(\delta(S))}{\min\{|S|, |\bar{S}|\}},$$

where $\bar{S} = V \setminus S$. The sparsity of the graph $G$ is defined as the sparsity of the sparsest cut,

$$\Phi(G) \triangleq \min_{S \subset V} \Phi(S).$$

To relate sparsity to the Laplacian, note that for any nonempty set $S \subset V$ with at most $n/2$ vertices, we have

$$\Phi(S) = \frac{\bar{w}(\delta(S))}{|S|} = \frac{\langle \mathbb{1}_S, L\mathbb{1}_S \rangle}{\langle \mathbb{1}_S, \mathbb{1}_S \rangle},$$

where $\mathbb{1}_S$ is the $\{0, 1\}$-indicator vector for $S$.

Now we make a deeper connection to the eigenvectors of $L$. As the positive semi-definite matrix $L$, $L$ has nonnegative eigenvalues. Moreover, we know that $1$ is an eigenvector with eigenvalue of 0 – this gives us our smallest eigenvalue. The eigenvector corresponding to the second smallest eigenvalue, denoted $\lambda_2$, is given by

$$\lambda_2 = \min_{x : \langle 1, x \rangle = 0} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.$$
Now, consider any cut \( 1_S \), and let \( x \) be the orthogonal projection from \( 1 \); namely,
\[
x = 1_S - \alpha 1 = \frac{\langle 1_S, 1 \rangle - \langle 1, 1 \rangle}{\langle 1, 1 \rangle} = \frac{|S|}{n}
\]
Observe that
\[
\langle x, x \rangle = \langle x, 1_S \rangle = (1 - \alpha) |S| = (n - |S|) |S| / n.
\]
Thus
\[
\lambda_2 \leq \frac{\langle x, Lx \rangle}{\langle x, x \rangle} = \frac{n \sum_{e \in \delta(S)} w(e)}{|S|(n - |S|)} \leq 2 \Phi(S).
\]
Taking the minimum over all sets \( S \), we obtain the following.

**Theorem 17.2.** Let \( G = (V, E) \) be an undirected graph with \( m \) edges and \( n \) vertices, and positive edge weights \( w : E \rightarrow \mathbb{R}_{>0} \). Let \( L : \mathbb{R}^V \rightarrow \mathbb{R}^V \) be the Laplacian of \( G \). Let \( \lambda_2 \) be the second smallest eigenvalue of \( L \) and let \( \Phi(G) \) be the sparsity of \( G \). Then
\[
\lambda_2 \leq n \Phi(G).
\]

### 17.4 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. Similarly to sparsity, we define the conductance of a graph as the minimum conductance of any cut:
\[
\Psi(G) = \min_{\emptyset \subsetneq S \subseteq V} \Psi(S).
\]
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
Figure 17.1: Examples of graphs with varying levels of sparsity and conductance.

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 $\text{vol}(S)$. To continue this analogy, the conductance of a (small) set $S$ models the amount of probability mass that enters and leaves $S$ in each step at the stationary distribution. Fig. 17.1 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 (17.1). While the numerator in (17.1) seems appropriate, the denominator does not capture the volume. Instead, consider the following quotient:

$$\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} \text{ where } x \in \mathbb{R}^V. \quad (17.2)$$

For any set $S$ with at most half the total volume, we have

$$\Psi(S) = \frac{\bar{w}(\delta(S))}{\text{vol}(S)} = \frac{\langle 1_S, L1_S \rangle}{\langle 1_S, D1_S \rangle}.$$ 

That said, the quotient (17.2) 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

$$M \overset{\text{def}}{=} D^{-1/2}LD^{-1/2}.$$ 

For any vector $x$, letting $y = D^{1/2}x$, we have

$$\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} = \frac{\langle y, My \rangle}{\langle y, y \rangle}.$$ 

236
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 conductance 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

$$Q \overset{\text{def}}{=} D^{-1/2}RD^{1/2} = D^{-1/2}AD^{1/2}.$$ 

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 17.3.** 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 $M : \mathbb{R}^V \to \mathbb{R}^V$ 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 17.4.** Let $M$ be the normalized Laplacian of an undirected graph $G$, and let $\lambda_2$ 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, and leads to a lot of tricky situations. However, for the sake of constant degree expanders – where the conductance equals the sparsity up to a constant, and where we are interested in constant sparsity/conductance – Cheeger’s inequality implies that the expansion and $\lambda_2$ are within a constant. An algorithmic proof of theorem 17.4 will give the verification algorithm we seek. We note that the LHS is more straightforward than the RHS, and left to the reader in exercise 17.3. We will prove the harder inequality momentarily in section 17.5.
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 smallest eigenvalue of the random walk matrix when this value is very close to $-1$. So instead we analyze the closely related lazy random walk where this exception does not occur.

**Theorem 17.5.** Let $G$ be an undirected graph with conductance $\Psi$ 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

$$\|S^t x - s\| \leq \exp\left(-t\Psi^2/4\right)\sqrt{\frac{\Delta_{\text{max}}}{\Delta_{\text{min}}}}$$

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 $k$.\(^2\) In particular, since $\mu_n(S) \geq -1/2$, $S$ has spectral gap

$$\gamma_S = \min\{1 - \mu_2(S), 1 + \mu_n(S)\} \geq 1/2 \min\{1 - \mu_2(R), 1\}. $$

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. \(\square\)

\(^2\)Here $\mu_k(A)$ denotes the $k$th largest eigenvalue of $A$, and $\lambda_k(A)$ denotes the $k$th smallest eigenvalue of $A$. 
17.5 Fiedler’s algorithm: proving the hard part of the Cheeger inequality

In this section we present an algorithmic proof of the upper bound, \( \Psi(G) \leq \lambda \), due to Fiedler. Beyond the surprising connection to the eigenvalues of \( M \), the algorithm is fairly simple, and based on previous discussions, the reader might be able to guess it.

Recall that \( \lambda_2 = \min_{y : \langle \sqrt{d}, y \rangle = 0} \frac{\langle y, My \rangle}{\langle y, y \rangle} = \min_{x : \langle d, x \rangle = 0} \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} \).

Let \( x \in \mathbb{R}^V \) with \( \langle d, x \rangle = 0 \) attain \( \lambda_2 \) 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 losing too much on the conductance. How?

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.

The answer given here is to output the best cut along the line embedding \( x \) – yet again! This is called Fiedler’s algorithm and pseudocode is given below.

1. Let \( x \) minimize \( \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} \) s.t. \( \langle x, d \rangle = 0 \) and \( x \neq 0 \).
2. For \( t \in \mathbb{R} \), let \( S_t = \{ v \in V : x_v \geq t \} \). Return the set \( S_t \) of minimum conductance.

As with \((s,t)\)-cut and sparsest cut before, the analysis is probabilistic. We will show that a randomized variant returns a satisfying cut with nonzero probability. For an appropriate distribution of random cuts alone the line embedding \( x \), we show that a random cut is good in expectation. More precisely, for \( t \in \mathbb{R} \), let \( S_t = \{ v \in V : x_v \geq t \} \).

We will find a distribution over \( t \in \mathbb{R} \) such that a random cut \( S_t \) has nonzero chance of good sparsity. The distribution will be more involved then the simple distribution for \((s,t)\)-cut, which was simply the uniform distribution.

**Recentering.** Before defining the distribution, it is convenient to recenter \( x \) so that 0 is a weighted median with respect to degrees. Explicitly, let \( y = x - \mu \mathbb{1} \) where \( \mu \in \mathbb{R} \) is a weighted median with respect to the degrees; i.e., \( \sum_{v : x_v \leq \mu} d_v \geq W \) and \( \sum_{v : x_v \geq \mu} d_v \geq W \) where \( W \) is the total sum of edge weights. Note \( y \) is not orthogonal to \( d \). But the following lemma shows that the affine shift by \( \mu \mathbb{1} \) only decreases the Rayleigh quotient. This is simply because translation by \( \mathbb{1} \) does not effect the numerator, and can only increase the denominator.
Lemma 17.6. Let $\langle x, d \rangle = 0$. For any $\alpha \in \mathbb{R}$,

$$\frac{\langle x + \alpha 1, L(x + \alpha 1) \rangle}{\langle x + \alpha 1, D(x + \alpha 1) \rangle} \leq \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle}.$$ 

Proof. We have $\langle x, Lx \rangle = \langle x + \alpha 1, L(x + \alpha 1) \rangle$ because $1 \in \ker(L)$. Meanwhile, consider the function

$$f(\alpha) = \langle x + \alpha 1, D(x + \alpha 1) \rangle \overset{(a)}{=} \langle x, Dx \rangle + \alpha^2 \langle d, 1 \rangle,$$

where for (a) we recall that $\langle d, x \rangle = 0$. Of course the RHS is minimized at $\alpha = 0$. 

The distribution. Let $t_0 = \min_v y_v$ be the smallest coordinate value in $y$ and let $t_1 = \max_v y_v$ be the largest coordinate value. Scaling $y$ if necessary, we assume that $t_0^2 + t_1^2 = 1$. The distribution over $t \in [t_0, t_1]$ is defined by

$$P[\alpha \leq t \leq \beta] = \alpha^2 + \beta^2$$

for $t_0 \leq \alpha \leq 0 \leq \beta \leq t_1$. For $t \in \mathbb{R}$, let $S_t = \{ v : y_v \leq t \}$. We will show that with nonzero probability, $\Psi(S_t) \leq \sqrt{2\lambda_2}$.

17.5.1 High level overview of the proof

In this section, we give a high-level overview of the analysis. In particular, we isolate two key lemma’s, and then use them to prove the overall theorem. These lemma’s are proven in subsequent sections.

The overall structure is similar to our previous discussion on sparsest cut. There we had a randomized ratio and we analyzed the expected value of the numerator and denominator separately. These were then combined to show that there exists a good cut. Similarly we start with a lemma addressing the numerator; i.e., the expected weight of edges cut by $S_t$.

Lemma 17.7. $E[\bar{w}(S_t)] \leq \sqrt{2\langle y, Ly \rangle \langle y, Dy \rangle}$.

Note the $\sqrt{\cdots}$ on the RHS, which will lead to the $\sqrt{\cdots}$ term in the final bound. The next lemma addresses the denominator; i.e., the volume on the smaller side of the cut.

Lemma 17.8. $E[\min \{ \text{vol}(S_t), \text{vol}(\bar{S}_t) \}] = \langle y, Dy \rangle$. 

\footnote{We note that specifying the probability of $t$ being in any given closed interval suffices to describe a continuous distribution $t \in \mathbb{R}$. Note that the rescaling sets $P[t_0 \leq t \leq t_1] = 1$.}
We now have bounds on the expectation of the numerator and denominator of $\Psi(S_t)$. As we showed in our discussion of sparsest cut, this implies there exists a cut $S_t$ whose conductance matches these expected values.

**Lemma 17.9.** $P[\Psi(S_t) \leq \frac{E[\bar{w}(S_t)]}{E[\min\{\text{vol}(S_t),\text{vol}(-S_t)\}]}] > 0.$

All put together, we obtain the following.

**Theorem 17.10.** In polynomial time, Fiedler’s algorithm computes $S$ such that $\Psi(S) \leq 2\sqrt{\Psi(G)}$.

**Proof.** We have

$$P\left[\Psi(S_t) \leq \sqrt{\frac{2\langle y, Ly \rangle}{\langle y, Dy \rangle}} \right] \geq P\left[\Psi(S_t) \leq \frac{E[\bar{w}(S_t)]}{E[\min\{\text{vol}(S_t),\text{vol}(-S_t)\}]} \right] > 0,$$

as desired. Here (a) is by lemmas 17.7 and 17.8 and (b) is by lemma 17.9.

### 17.5.2 Expected size of the cut

We first analyze the expected size of the random cut. The first probability gives an upper bound on the problem of an edge $e = \{u, v\}$, in terms of the coordinates $y_u$ and $y_v$.

**Lemma 17.11.** For an edge $e = \{u, v\} \in E$,

$$P[e \in \delta(S_t)] \leq |y_u - y_v|(|y_u| + |y_v|).$$

**Proof.** We have three different cases, depending on where $y_u$ and $y_v$ lie relative to $\mu$. We assume without loss of generality that $y_u \leq y_v$.

**Case 1.** Suppose $y_u \leq 0 \leq y_v$.

Then

$$P[e \in \delta(S_t)] = P[y_u \leq t \leq y_v] = y_u^2 + y_v^2 \overset{(a)}{=} |y_u - y_v|(|y_u| + |y_v|).$$

where (a) observes that $|y_u| \leq |y_u - y_v|$ and $|y_v| \leq |y_u - y_v|$.  

241
Case 2. Suppose \( 0 \leq y_u \leq y_v \)

Then
\[
P[e \in \delta(S_t)] = y_v^2 - y_u^2 = (|y_v| + |y_u|)(|y_v| - |y_u|) \leq |y_v - y_u|||y_v| + |y_u||,
\]
where (b) observes that \( |y_v| = y_v, |y_u| = y_u, \text{ and } y_v - y_u = |y_v - y_u| \).

Case 3. Suppose \( y_u \leq y_v \leq 0 \).

Then
\[
P[e \in \delta(S_t)] = y_v^2 - y_u^2 = (|y_u| + |y_v|)(|y_u| - |y_v|) \leq |y_v - y_u|||y_v| + |y_u||,
\]
where (c) observes that \( |y_v| = y_v, |y_u| = y_u, \text{ and } y_v - y_u = |y_v - y_u| \).

The next lemma bounds the expected weight of the cut.

**Lemma 17.7.** \( \mathbb{E}[\bar{w}(S_t)] \leq \sqrt{2\langle y, Ly \rangle \langle y, Dy \rangle} \).

**Proof.** We have,
\[
\mathbb{E}[\bar{w}(S_t)] = \sum_{\{u, v\} \in E} w(e) P[e \in \delta(S)] \\
\leq \sum_{\{u, v\} \in E} w(e)|y_v - y_u|(|y_v| + |y_u|) \\
\leq \left(\sum_{\{u, v\} \in E} w(e)(y_v - y_u)^2\right)^{\frac{1}{2}} \sqrt{\sum_{\{u, v\} \in E} w(e)(|y_v| + |y_u|)^2}.
\]
by (a) Cauchy-Schwartz. For the first term, we have
\[
\sum_{\{u, v\} \in E} w(e)(y_v - y_u)^2 = \langle y, Ly \rangle.
\]
For the second term, we have
\[
\sum_{\{u, v\} \in E} w(e)(|y_v| + |y_u|)^2 \leq 2 \sum_{\{u, v\} \in E} w(e)(y_v^2 + y_u^2) = 2 \sum_u \text{vol}(u)y_u^2 = \langle y, Dy \rangle,
\]
where (b) applies the inequality \( 2ab \leq a^2 + b^2 \).
17.5.3 Expected volume

It remains to analyze the expected volume, as follows.

Lemma 17.8. \( E[\min\{\text{vol}(S_t), \text{vol}(\bar{S}_t)\}] = \langle y, Dy \rangle \).

Proof. If \( t \geq 0 \), then \( \text{vol}(S_t) \leq \text{vol}(\bar{S}_t) \), and if \( t \leq 0 \), then \( \text{vol}(\bar{S}_t) \leq \text{vol}(S_t) \). We have \( P[t \geq 0] = (t_1)^2 \) and

\[
E[\text{vol}(S_t) \mid t \geq 0] = \frac{1}{t_1^2} \sum_{y_v \geq 0} d_v^2 y_v^2.
\]

Similarly, we have \( P[t \leq 0] = t_0^2 \) and

\[
E[\text{vol}(\bar{S}_t) \mid t \leq 0] = \frac{1}{t_0^2} \sum_{y_v \leq 0} d_v^2 y_v^2.
\]

By conditioning on whether \( t \) is \( \geq 0 \) or \( \leq 0 \), we have

\[
E[\min\{\text{vol}(S_t), \text{vol}(\bar{S}_t)\}] = E[\text{vol}(S_t) \mid t \geq 0] P[t \geq 0] + E[\text{vol}(S_t) \mid t \leq 0] P[t \leq 0]
= \sum_{y_v} d_v^2 y_v^2 = 2\langle y, Dy \rangle,
\]

as desired. \( \square \)

Additional notes and references

The proof here is based on presentations in [trevisan-spectral-notes; spielman], which we refer the reader to for more on spectral graph theory.

17.6 Exercises

Exercise 17.1. 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 \( G \). Prove that \( G \) is connected iff for any \( x \notin \text{span}(\mathbf{1}) \), we have \( \langle x, Lx \rangle > 0 \).

Exercise 17.2. Finish the proof of lemma 16.3, 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 17.3. Here we prove the lower bound in Cheeger’s inequality (theorem 17.4), \( \Psi(G) \geq \lambda_2/2 \). 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 \( x = D^{1/2}1_S \) and let \( y = D^{1/2}1_{\overline{S}} \).

1. Show that
\[
\frac{\langle x, Mx \rangle}{\langle x, x \rangle} = \frac{\langle 1_S, L1_S \rangle}{\langle 1_S, D1_S \rangle}, \quad \frac{\langle y, My \rangle}{\langle y, y \rangle} = \frac{\langle 1_{\overline{S}}, L1_{\overline{S}} \rangle}{\langle 1_{\overline{S}}, D1_{\overline{S}} \rangle}, \text{ and } \langle x, y \rangle = 0.
\]

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 \( \langle d, \alpha x + \beta y \rangle = 0 \).

4. Finally, prove that the second smallest eigenvector of \( M \) is at most \( 2\Psi(G) \).
Chapter 18

Deterministic log-space connectivity

18.1 Introduction

Many of our recent discussions have been about random walks in graphs. We have seen that random walks are both useful and, with the help of some mathematical tools, analyzable as well. Today’s discussion is titled deterministic walks. Compared to random walks, there would seem to be no mystery or potential or interest in deterministic walks. Surely we all know how to deterministically walk around a graph. Thus let us clarify, or rather rephrase, our topic of discussion to derandomizing random walks.

More precisely, 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 instead we asked if one could do it in logarithmic space. In particular we cannot mark the vertices as we visit them. Surprisingly we could decide reachability in logarithmic space, by randomly walking around from \(s\) and answering yes if we reach \(t\) within the first \(O(mn)\) steps. This was connected to the cover time: via an unexpected detour through electrical networks, we showed that the expected number of steps until a random walk from \(s\) visits every vertex (connected to \(s\)) is \(\leq 2mn\). A follow up question in the homework showed that if \(s\) and \(t\) are connected by a path of \(k\) edges, then the expected number of steps is \(O(mk)\).

We have also established, in previous discussions, 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, we showed that the convergence rate is connected to the spectral gap \(\gamma\) of the random walk matrix - that 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 \(d\). Since the sum of degrees is \(2m\), the stationary distribution is \(d/2m\). 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 \( s \) at the rate of

\[
\|x_k - \frac{d}{2m}\| \leq (1 - \gamma)^k n.
\]

To help interpret the Euclidean norm, keep in mind that \( \|x_k - \frac{d}{2m}\|_\infty \leq \|x_k - \frac{d}{2m}\| \).

If the spectral gap \( \gamma \) is a constant - say, for the sake of discussion, \( \gamma = 1/2 \) - then we converge at an exponentially fast rate:

\[
\|x_k - \frac{d}{2m}\| \leq \frac{n}{2^k}.
\]

For \( k = O(\log n) \) steps, we have

\[
\|x_k - \frac{d}{2m}\|_\infty \leq \|x_k - \frac{d}{2m}\| \leq \frac{n}{2^k}.
\]

Since \( d(v)/2m \geq 1/2m \) for all \( v \), we have \( x_k(v) > 0 \) for \( k \geq O(\log n) \). Note that this holds for any initial distribution. Let’s say we started a single vertex \( s \). Then \( x_k(v) > 0 \) implies, in particular, that there exists a path from \( s \) to \( v \) of length \( k \). That is, any graph with constant spectral graph has \( O(\log n) \) diameter. In the context of our \((s,t)\)-connectivity,

To push this thought experiment further, suppose also that \( G \) had maximum degree at most a constant, say, 10. In particular, it only takes a constant number of random bits to make each random step. If there is a path from \( s \) to \( t \) of length \( O(\log n) \) steps, then there are \( O(\log n) \) random bits that tell us how to get there. A random walk is like guessing these random bits. But when there’s only \( O(\log n) \) bits to guess, we can enumerate and try all \( 2^{O(\log n)} = \text{poly}(n) \) possible bit strings!

**Observation 18.1.** \((s,t)\)-connectivity in a constant degree expander can be decided deterministically in \( O(\log n) \) space.

Our goal is to prove the following theorem due to reingold. It asserts that we can deterministically decide \((s,t)\)-connectivity in logarithmic space on any graph, not just expanders.

**Theorem 18.2 ([reingold]).** There is a \( O(\log n) \) space, polynomial time deterministic algorithm for \((s,t)\)-connectivity in undirected graphs with \( n \) vertices.
18.2 An overview of the deterministic connectivity algorithm

We first give a high level of the algorithm and analysis of theorem 18.2. We will identify some key technical lemma’s, and assume them for the moment and give the rest of the proof theorem 18.2. We will prove the lemma’s in subsequent sections.

The approach taken by [reingold] starts from Observation 18.1 that deterministic log-space connectivity is easy on expander graphs. While of course not all graphs are expanders, [reingold] applied a sequence of graph transformations (building on previous work such as [rvw]) that turn the input graph (implicitly) into an expander. We can then deterministically simulate the random walk algorithm on this expander by enumeration.

18.2.1 Two operations transformations on graphs

**Powering**

Let $G = (V, E)$ be an undirected graph with $m$ edges and $n$ vertices. For $k \in \mathbb{N}$, the $k$th power is the multi-graph on $V$ generated by 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 $v$.

Observe that if $R$ is the random walk matrix on $G$, then $R^k$ is the random walk matrix on $G^k$. Recall from before that taking multiple steps along a random walk amplifies the spectral gap. Intuitively, powering improves the spectral gap, but increases the degree.

**Lemma 18.3.** 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 $\gamma$, then $R^k$ has spectral gap $1 - (1 - \gamma)^k$.

The proof of the lemma is implicit in our previous discussion on convergence rates on undirected graphs.

**Zig-Zag Product**

Our second operation operation is called the *zig-zag product*. The goal of the zig-zag product is to reduce the degree of the graph.
Let $G = (V, E)$ be a regular undirected graph $n$ vertices and degree $d$. The high level goal is to reduce the degree $d$. Let $H = (V_0, E_0)$ be a regular undirected graph with $d$ vertices and degree $d_0$. Morally, $d_0$ is a universal constant. Note that the number of vertices in $H$ matches the degree of $G$ exactly.

Each vertex in $H$ can be interpreted as a choice of neighbor in $G$. Speaking very informally, rather than walking on $G$ directly (which requires $\log(d)$ bits per step), we walk on $H$ (which requires only $\log(d_0)$ bits per step), and use the locations in $H$ to induce a walk on $G$. In this spirit, we identify the vertex of $H$ with the set of indices $[d] = \{1, \ldots, d\}$.

The zig-zag product, denoted $Z(G \mid H)$, is a regular graph with vertex set $V_1 \times V_2$ and degree $d^2$. Each edge in $Z(G \mid H)$ 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). For precisely, let $(v_1, i_1) \in V_1 \times V_2$. 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.

\[
\begin{align*}
(v_1)_{i_1} \to (v_1)_{i_2} \to (v_2)_{i_2} \to (v_2)_{i_3} \to (v_2)_{i_4}
\end{align*}
\]

(a) is a step in $H$ from $i_1$ to its $k_1$th neighbor, $i_2$. (b) moves $v_1$ to the $i_2$th outgoing neighbor of $v_1$, $v_2$. (c) uses $v_1$ and $v_2$ to move from $i_2$ to $i_3$; namely, we move from $i_2$ to $i_3$ if $v_1$ is the $i_3$th incoming neighbor of $v_2$. (d) moves $i_3$ to its $k_2$th neighbor in $H$.

Note that in the move from $(v_1, i_1)$ to $(v_2, i_4)$, $v_1$ and $v_2$ are neighbors in $G$. However, $i_1$ and $i_4$ are not (necessarily) neighbors in $H$. 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 $i_3$. Fortunately, 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 $Z(G \mid H)$, then $s$ and $t$ are connected in $G$.

Intuitively, the zig-zag product uses a walk on $H$ to implicitly define a constant degree walk on $G$. In subsequent applications, $H$ will have constant spectral gap, which intuitively means that a random step in $H$ is close to a uniformly random vertex in $v$.

We defer the proof of the following lemma until section 18.3.

**Lemma 18.4.** Let $G = (V, E)$ be a regular undirected graph with $n$ vertices and degree $d$, with spectral gap $\gamma_G$. Let $H$ be a regular undirected graph with $d$ vertices and degree $d_0$. Then $Z(G \mid H)$ is a regular undirected graph with $nd$ vertices, degree $d_0^2$ and spectral gap $\gamma_G \gamma_H^2$. 
18.2.2 Completing the proof

**Lemma 18.5.** Let $G$ be a regular undirected graph with $n$ vertices, degree $d^2$, and spectral gap $\gamma_G$. Let $H$ be a regular undirected graph with $d^4$ vertices, degree $d$, and spectral gap $\gamma_H$. Then $Z(G^2 \mid H)$ is a regular undirected graph with $d^4n$ vertices, degree $d^2$, and spectral gap $(1 - (1 - \gamma_G)^2)\gamma_H^2$.

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 $\geq \frac{1}{\text{poly}(n)}$. We can also assume that there exists a regular expander $H$ on $d^4$ vertices, degree $d$, and spectral gap $\geq \frac{3}{4}$. If we apply lemma 18.5 to $\gamma_G \leq \frac{1}{16}$ and $\gamma_H \geq \frac{3}{4}$, then $Z(G^2 \mid H)$ 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 $d^2$ – a constant.

We need to check that the number of vertices did not blow up too much. Each time we apply lemma 18.5, the number of vertices increases by $d^4$. After $O(\log n)$ iterations – if $d$ is a constant – the number of vertices increases by at most $\text{poly}(n)$. Thus we have a constant degree graph with $\text{poly}(n)$ vertices and constant spectral-gap – primed for derandomizing the random walk approach.

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 $G_k = Z(G^2_{k-1} \mid H)$ be the graph obtained by the $k$th iteration of lemma 18.5. We claim the following for each index $j$.

1. The space required to simulate a step on $G_j^2$ is $O(1)$ plus the space required to simulate a step on $G_j$.
2. The space required to simulate a step on $Z(G_j^2 \mid H)$ 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, were we are simulate a step on $G_j^2$. We are given a vertex $v_1$ in $G_j^2$ and two indices $i_1, i_2 \in [d]$. 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 $G_j$. 

307
Consider the second claim, where we are simulating a step on $Z(G_j^2 | H)$. 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 $j_1, j_2 \in [d_0]$. We first take a step in $H$ from $i_1$ to $i_2$, using $O(1)$ space. We then query $G_j^2$ to take $i_2$th edge from $v_1$ to $v_2$ in $G_j^2$. We then query, for each $i_3 \in [d]$, the $i_3$th edge from $v_2$ in $G_j^2$ until we find that $v_1$ is the $i_3$th 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 $G_j^2$.

Up to proving lemma 19.8 in the subsequent section, this completes the proof of theorem 18.2.

18.3 Analysis of the zig-zag product

**Lemma 19.8.** Let $G = (V, E)$ be a regular undirected graph with $n$ vertices and degree $d$, with spectral gap $\gamma_G$. Let $H$ be a regular undirected graph with $d$ vertices and degree $d_0$. Then $Z(G | H)$ is a regular undirected graph with $nd$ vertices, degree $d_0^2$ and spectral gap $\gamma_G \gamma_H^2$.

Let $R_H$ be the random walk matrix of $H$. We can write a step in the zig-zag graph as

$$(I \otimes R_H)Z(I \otimes R_H),$$

where $(I \otimes R_H)$ represents the action where we take a single random step in $H$ but leave the $G$-coordinate fixed. $(I \otimes R)$ also describes a random step in a certain “tensor product” of graphs that we will more formally discuss in a moment. $Z$ 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 in our proof, $H$ will be an expander. That is, taking a few steps in $R_H$ is almost as random as sampling a uniformly random vertex from $H$. To this end, let $S : \Delta^V_H \rightarrow \Delta^V_H$ be the “random walk” that models sampling a vertex from $H$ uniformly at random in each step. 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 a different graph (say) $H'$, which is a complete graph with a self-loop at every vertex. Let us walk through a random step in the zig-zag product $\mathcal{Z}(G \mid H')$.

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 $i_2$th neighbor $v_2$.

3. Then, we move from $i_2$ to $i_3$ where $v_1$ is the $i_3$th 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 $H'$. 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 is precisely $\gamma_G$, the spectral gap of $G$.

Of course, $S$ is not exactly $R_H$, and 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 $H$, $\gamma_H$, is also a reflect of the difference between $R_H$ and $S$. This difference between $R_H$ and $S$, and the correspondance between the difference and $\gamma_H$, is why the spectral gap decreases from $\gamma_G$ to $\gamma_G \gamma_H^2$.

18.3.1 Introducing tensor products

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. The tensor product of $G_1$ and $G_2$, denoted $G_1 \otimes G_2$, is the graph defined as follows. The vertex set is the family $V_1 \times V_2$ of all pairs of vertices from $V_1$ and $V_2$,

$$V_1 \times V_2 = \{(v_1, v_2) : v_1 \in V_1, v_2 \in V_2\}.$$ 

For each edge $e_1 = (u_1, u_2)$ and each edge $e_2 = (v_1, v_2)$, we have an edge $((u_1, u_2), (v_1, v_2))$ in $G_1 \otimes G_2$, with multiplicities. That is, an edge $((u_1, u_2), (v_1, v_2))$ in $G_1 \otimes G_2$ has multiplicity equal to the multiplicity of $(u_1, v_1)$ in $E_1$ times the multiplicity of $(u_2, v_2)$ in $E_2$. 

251
Theorem 18.6. Let $G_1$ and $G_2$ be regular undirected graphs with random walk
matrices $R_1$ and $R_2$ respectively. 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 correspondence between pairs of eigenvectors from
$G_1$ to $G_2$, where an eigenvector $v_1$ with eigenvalue $\lambda_1$ of $G_1$ and an eigenvector $v_2$
of $G_2$ maps to an eigenvector $v_1 \otimes v_2$ of $G_1 \otimes G_2$ with eigenvalue $\lambda_1 \lambda_2$.

Proof. We have already remarked that the tensor product of two regular vertices
is again regular, so the corresponding random walk matrix $R_1 \otimes R_2$ is regular. Let $u_1, \ldots, u_n$
be $n$ orthonormal eigenvectors of $R_1$ and let $v_1, \ldots, v_n$ be the $n$ orthonormal
eigenvectors of $R_2$. Then by direct inspection one can show that (a) the family
$\{u_i \otimes v_j : i \in [n_1], j \in [n_2]\}$ are orthonormal in $\mathbb{R}^{V_1 \times V_2}$ and (b) each $u_i \otimes v_j$
is an eigenvector of $R_1 \otimes R_2$ with eigenvalue $\lambda_i \lambda_j$. \qed

18.3.2 Tensors with the identity

Recall that the following tensor product in particular arises in particular in the zig-zag
product $\mathcal{Z}(G | V)$:

$$I \otimes R_H.$$ 

We can think of this as a random step of the following tensor product. Let $G_0$ be the
graph with the same vertex set as $G$ and no edges except for a self-loop at each step.
Then a “random step” on $G_0$ goes no where at all - we stay on the same vertex. In
particular, the random walk matrix is $I$ – the identity map on $V$.

Consider the tensor product $(G_0 \otimes H)$. Each random step on $(G_0 \otimes H)$ consists of
a “random” step on $G_0$ - which just keeps the same coordinate in $V$ - and a random
step on $H$, for which we denote the random walk matrix by $R_H$. Thus

$$I \otimes R_H,$$

which was was introduced as taking a random step in $H$ and staying put in $G$ - is
precisely the random walk matrix.

While tensor products of linear maps are more generally defined, they are easier
to describe concretely in the special case where one of the maps is the identity map.
Take for example $I \otimes R_H$. Given a vector $x \in \mathbb{R}^{V \times d}$, let $x_u \in \mathbb{R}^d$ denote the “$u$-slice”
of $x$ defined by

$$(x_u)(i) = x(u, i).$$

252
18. Deterministic log-space connectivity

18.3. Analysis of the zig-zag product

We can think of \( I \otimes R_H \) as applying \( R_H \) to each slice; i.e.,

\[
((I \otimes R_H)x)_u = R_Hx_u.
\]

More generally, if \( I : \mathbb{R}^{n_1} \to \mathbb{R}^{n_1} \) is the identity map, \( A : \mathbb{R}^{n_2} \to \mathbb{R}^{n_2} \) is any linear map, and \( i \in [n_1] \), then we have

\[
((I \otimes A)x)_i = Ax_i.
\]

Form this definition we get the following useful algebraic rule:

\[
(I \otimes A + B) = (I \otimes A) + (I \otimes B),
\]

since

\[
((I \otimes A + B)x)_i = (A + B)x_i = Ax_i + Bx_i = ((I \otimes A)x)_i + ((I \otimes B)x)_i.
\]

18.3.3 A helpful inequality

Lemma 18.7. The operator norm

\[
\|A\| = \sup\{\|Ax\| : \|x\| = 1\}.
\]

Lemma 18.8. Let \( A : \mathbb{R}^n \to \mathbb{R}^n \) be a symmetric and positive semi-definite linear map, and let \( \lambda \geq 0 \) be the maximum absolute value of any eigenvalue of \( A \). Then

\[
\|A\| = \lambda.
\]

Proof. Recall that

\[
\lambda = \max_x \left\| \frac{\langle x, Ax \rangle}{\langle x, x \rangle} \right\|,
\]

and by the spectral theorem,

\[
\lambda^2 = \max_x \frac{\langle x, A^2x \rangle}{\langle x, x \rangle}.
\]

On the other hand,

\[
\|A\|^2 = \sup_{\|x\|=1} \langle Ax, Ax \rangle = \sup_{\|x\|=1} \langle x, A^2x \rangle = \lambda^2.
\]

Lemma 18.9. Let \( A \) be a random walk matrix with stationary distribution \( u \). Then for any non-dominant eigenvalue \( \lambda \neq 1 \), we have

\[
\lambda \leq \|A\|.
\]
18.3.4 Zig-Zags and tensor products

Let $R_H$ be the random walk matrix of $H$. Recall previously that we applied the spectral theorem to $R_H$, indirectly by similarity to the normalized random walk matrix, to derive very precise bounds on the converge rate towards a random walk, as a function of the spectral gap. Here to we want to analyze the eigenvalues of the zig-zag product, and we start from reanalyzing $R_H$. In some ways our analysis is even easier this time because when $G$ is a regular graph with constant degree $d$, $R_H = \frac{1}{d} A$ is already a symmetric map! (Here $d$ is an integer, and not a vector). 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 $u_1, \ldots, u_n \in \mathbb{R}^d$ forms an orthonormal bases and $\lambda_2, \ldots, \lambda_n \in [1 - \gamma_H; \gamma_H - 1]$. Recall that because $R_H$ is regular, the stationary [distribution is the uniform distribution, $\frac{1}{d}$. Since $R_H \mathbb{1} = \mathbb{1}$, the first eigenvector $u_1$ must be (proportional to) $\mathbb{1}$. This gives

$$R_H = \frac{1}{d} (\mathbb{1} \otimes \mathbb{1}) + \lambda_2 (u_2 \otimes u_2) + \cdots + \lambda_n (u_n \otimes u_n),$$

Consider the first term $\frac{1}{d} (\mathbb{1} \otimes \mathbb{1})$. This map sends any distribution to the stationary one, $\frac{1}{d}$ – that is, this is exactly the map $S$! Thus we have

$$R_H = \frac{1}{d} (\mathbb{1} \otimes \mathbb{1}) + \lambda_2 (u_2 \otimes u_2) + \cdots + \lambda_n (u_n \otimes u_n),$$

where $\frac{1}{\sqrt{d}}, u_2, \ldots, u_n$ form an orthonormal basis and $\lambda_2, \ldots, \lambda_n \in [1 - \gamma_H; \gamma_H - 1]$. Let us decompose $R_H$ as

$$R_H = S + T$$

where $S = \frac{1}{d} (\mathbb{1} \otimes \mathbb{1})$ and $T = \lambda_2 (u_2 \otimes u_2) + \cdots + \lambda_n (u_n \otimes u_n)$. Let

$$R'_H = R_H - \gamma_H S;$$

then $R'_H$ has all its eigenvalues in the range $[1 - \gamma_H, \gamma_H - 1]$. Let us now return to the zig-zag product and substitute in for $R_H$. We have

$$(I \otimes R_H)Z(I \otimes R_H) = (I \otimes \gamma_H S + R'_H)Z(I \otimes \gamma_H S + R'_H)$$

$$= (\gamma_H (I \otimes S) + (I \otimes R'_H))Z(\gamma_H (I \otimes S) + (I \otimes R'_H))$$

$$= \gamma_H^2 (I \otimes S)Z(I \otimes S)Z + \gamma_H ((I \otimes S)Z(I \otimes R'_H) + (I \otimes R'_H)Z(I \otimes S)) + (I \otimes R'_H)Z(I \otimes R'_H).$$

254
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 (which is the first eigenvector). We have

\[
\langle z, (I \otimes S)Z(I \otimes R'_H)z \rangle = \| (I \otimes S)Z(I \otimes R'_H)x \| \leq \| (I \otimes S) \| \| Z \| \| (I \otimes R'_H) \| \leq 1 - \gamma_H .
\]

Likewise the second term contributes \( 1 - \gamma_H \) (times another \( \gamma_H \)) and the third 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

\[
(I \otimes S)Z(I \otimes S) = (A \otimes S)
\]

Note that \((A \otimes S)\) 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

\[
\langle x, (I \otimes S)Z(I \otimes S)x \rangle = \langle x, (A \otimes S)x \rangle \leq 1 - \gamma_G.
\]

Putting everything together gives the desired bound. Viola!
Chapter 19

Derandomization by random walks

19.1 Introduction

Recall that a language $L$ is in the class $P$ if there is a deterministic $\text{poly}(n)$-time algorithm that decides if an input $x$ of size $n$ is in $L$. This notion has been to randomized algorithms as follows.

**Definition 19.1.** A language $L$ is in the class $\text{RP}$ 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 $\text{BPP}$ 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

$$P \subseteq \text{RP} \subseteq \text{BPP},$$

since of course, no error ($P$) is better than one-sided error ($\text{RP}$), which is better than two-sided error ($\text{BPP}$). It is a major open question if these are equal. Many believe that $P = \text{BPP}$. In practice, researchers treat a randomized algorithm with two-sided error is a strong indicator for existence of a deterministic one. Right now we do not know if $P$ equals $\text{RP}$ or if $\text{RP}$ equals $\text{BPP}$. 
There is an interest in the theory community, sometimes under the heading of pseudorandomness, in a refined understanding of how much randomness is required for certain problems. While the holy grail is $P$ vs $BPP$ is hard to attack, there is a rich body of literature or results moving towards this conclusion, that has also produced many algorithmic ideas of independent interest.

Let $L \in \text{RP}$, 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 $\delta$, for some $\delta$, then we could independently repeat the algorithm $\lceil \log 1/\delta \rceil$ times, taking the disjunction of responses. Often times we deemphasize the logarithmic overhead incurred from repetition, particularly since it is easily made up for by the convenience of randomization. Still it is a natural and profound question to ask if one can do better. Surprisingly, one can.

**Theorem 19.2.** Given an algorithm in $\text{RP}$ that uses $m$ random bits to achieve error $1/2$, one can decrease the error probability to $\delta$ while increasing the running time by a factor of $\log(1/\delta)$ and using a total of $m + O(\log(1/\delta))$ random bits.

We can do the same for algorithms in $\text{BPP}$. The algorithm will be the same as for $\text{RP}$, though the analysis is different.

**Theorem 19.3.** Given an algorithm in $\text{BPP}$ that uses $m$ random bits to achieve error $1/3$ (on both sides), one can decrease the error probability to $\delta$ while increasing the running time by a factor of $\log(1/\delta)$ and using a total of $m + O(\log(1/\delta))$ random bits.

### 19.2 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 it given.) Start from 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. For each vertex $v_i$ we visit, use $v_i$ as the input for a new instance of the algorithm. For algorithms in $\text{RP}$, take the disjunction (the “or”) of all the outputs. For algorithms in $\text{BPP}$, take the mean.

The above construction suggests that random bit strings generated by expanders are almost as good as completely random bit strings. We have two basic issues to address.

1. We need to show that the bit strings generated by randomly walking an expander are (probably) useful, for both the $\text{RP}$ and $\text{BPP}$ case.
2. We need to show how to efficiently make such a graph $G$.

Note that for the second point, since $G$ is exponential-size, this construction has to be implicit.

**Random walks for RP.** Let us first consider the first point - why bit strings generated by an expander graph on bit strings seems to be almost as good as totally random bit strings. We will have different lemma’s for either case. The first lemma is for amplifying RP.

**Lemma 19.4.** Let $G = (V, E)$ be a regular undirected graph whose random walk has spectral gap $\gamma$. 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 19.3. First let us reconsider theorem 19.2, for amplifying RP via expanders, in light of lemma 19.4. We apply lemma 19.4 to the expander graph of bit strings where we set $B$ to be the set of all random strings causing the algorithm to err. Amplifying the original algorithm a constant number of times as needed, we can make $\mu$ arbitrarily small; say, $1/2$. By lemma 19.4, the probability that all the bit strings we along the walk are bad drop at a rate or $(1 - \gamma/2)^t$.

**Random walks for BPP.** We now present the lemma that is important for BPP.

**Lemma 19.5.** 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 $f : V \rightarrow [0, 1]$ be a fixed function of the vertices. Let

$$\mu = \mathbb{E}[f(v)] \text{ where } v \in V \text{ uniformly at random.}$$

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 $\beta > 0$,

$$\mathbb{P}\left[\frac{1}{k} \sum_{i=1}^{k} f(v_i) - \mu \geq \epsilon \mu + \beta\right] \leq ce^{ck((1+\epsilon)\epsilon-\beta)}$$
for a universal constant $c > 0$. In particular, for (say) $\epsilon \leq \mu/4$, we have

$$P\left[\left|\frac{1}{k} \sum_{i=1}^{k} f(v_i) - \mu\right| \geq \epsilon \mu\right] \leq ce^{-\frac{\epsilon \mu}{c}}$$

for a universal constant $c > 0$.

Analogous to our discussion for RP, ?? implies theorem 19.3. We prove ?? in section 19.4.  

**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 19.5.

**Lemma 19.6.** Let $H$ be a graph with $d^8$ vertices, degree $d$, and spectral gap $\geq 7/8$. Define graphs $G_1, G_2, \ldots$ by

$$G_1 = H^2$$

$$G_{t+1} = Z\left((G_t \otimes G_t)^2 \mid H\right)$$

Let $n_t$ be the number of vertices in $G_t$. Then $n_t \approx c^{2t}$, and simulating one step of a random walk in $G_t$ takes $\text{poly}(\log(n_t))$ time.

### 19.3 Amplifying RP

**Lemma 19.4.** Let $G = (V, E)$ be a regular undirected graph whose random walk has spectral gap $\gamma$. 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 $B \subseteq V$, the probability that the entire random walk stays in $B$ is

$$(\mu + (1 - \gamma)(1 - \mu))^t,$$

where $\mu = |B|/|V|$.

**Proof.** Let $R$ be the random walk matrix on $G$. Let $P : \mathbb{R}^V \to \mathbb{R}^V$ be the projection subspace spanned by $\mathbb{R}^S$; that is,

$$(Px)_v = \begin{cases} x_v & \text{if } v \in S \\ 0 & \text{otherwise.} \end{cases}$$

Kent: Elaborate; in particular one should take a majority vote.
Note that $P$ is a linear function with $P = P^T$ and $P^2 = P$. 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 drop all the probability mass. In particular, the probability that the entire walk stays in $B$ is

$$\langle 1, (PRP)^t(1/n) \rangle.$$  

We claim that $PRP$ has maximum eigenvalue $\leq \mu + 1 - \gamma$. If so, then

$$\langle 1, (PRP)^t(1/n) \rangle = \frac{\langle 1, (PRP)^t1 \rangle}{\langle 1, 1 \rangle} \leq (\gamma \mu + 1 - \gamma),$$

which completes the proof.

Recall that

$$R = \frac{\gamma}{n} (1 \otimes 1) + R'$$

where $R'$ has all of its eigenvalues in the range $[1 - \gamma, \gamma - 1]$. Then

$$PRP = \frac{\gamma}{n} P(1 \otimes 1) P + PR'P.$$  

$PR'P$ has all its eigenvalues in the range $[1 - \gamma, \gamma - 1]$ too. Consider the first term. We claim that it has maximum eigenvalue $\gamma \mu = |S|/n$, which would give the overall bound of

$$\gamma \mu + 1 - \gamma = \mu + (1 - \mu)(1 - \gamma)$$

that we seek. To this end, we have

$$\frac{\gamma}{n} P(1 \otimes 1) P = \frac{\gamma}{n} (P1 \otimes P1)$$

and the maximum eigenvector of the outer product $(P1 \otimes P1)$ is (proportional to) $P1$ with eigenvalue

$$\frac{\langle P1, P1 \rangle^2}{\langle P1, P1 \rangle} = \langle P1, P1 \rangle = |S|.$$  

This gives the desired bound. \qed
19.4 Efficiently amplifying BPP

Lemma 19.5. 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

$$\mu = \mathbb{E}[f(v)] \text{ where } v \in V \text{ uniformly at random.}$$

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 $\beta > 0$,

$$\mathbb{P}\left[ \left| \frac{1}{k} \sum_{i=1}^k f(v_i) - \mu \right| \geq \epsilon \mu + \beta \right] \leq c e^{k(1 + \epsilon)\epsilon - \beta}$$

for a universal constant $c > 0$. In particular, for (say) $\epsilon \leq \mu/4$, we have

$$\mathbb{P}\left[ \left| \frac{1}{k} \sum_{i=1}^k f(v_i) - \mu \right| \geq \epsilon \mu \right] \leq c e^{-\epsilon k \mu}$$

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 the union bound over both. We have

$$\mathbb{P}\left[ \sum_{i=1}^k f(v_i) \geq k\mu + \beta \right] \leq \mathbb{E}\left[ e^{f(v_1) + \cdots + f(v_k)} \right] e^{-ek\mu - \epsilon \beta}. \quad (19.1)$$

The key identity is

$$\mathbb{E}\left[ e^{f(v_1) + \cdots + f(v_k)} \right] = \frac{1}{n} \langle 1, F(RF)^k 1 \rangle$$

where $F = \text{diag}(e^{f(v_1)}, \ldots, e^{f(v_n)})$ 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 $e^{f(v)}$ along the walk. We claim the following.

Claim 1. $\frac{1}{n} \langle F^{1/2} 1, F(RF)^k F^{1/2} 1 \rangle \leq e^{\epsilon (k(1 + \epsilon)\epsilon (\mu + \epsilon))}$
Assuming claim 1 holds, we have

\[(19.1) \leq e^\epsilon \cdot e^{\epsilon k((1+\epsilon)\epsilon - \beta)},\]

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 \(\epsilon\) and \(-\epsilon\). Thereby

\[F^{1/2}RF^{1/2} = \frac{1 - \epsilon}{n} F^{1/2}(\mathbb{1} \otimes \mathbb{1}) F^{1/2} + F^{1/2}R'F^{1/2}.\]

We claim the following.

Claim 2. \(F^{1/2}R'F^{1/2}\) has its eigenvalues between \([\epsilon \epsilon^f, -\epsilon \epsilon^f]\).

Claim 3. \(F^{1/2}(\mathbb{1} \otimes \mathbb{1})F^{1/2}\) has maximum eigenvalue \(\mathbb{E}[e^{\epsilon f(v)}]\).

For the first claim regarding \(F^{1/2}R'F^{1/2}\), for any vector \(x\)

\[\langle x, F^{1/2}R'F^{1/2}x \rangle = \langle F^{1/2}x, R'(F^{1/2}x) \rangle \leq \epsilon \|F^{1/2}x\|^2\]

\[= \epsilon \langle x, Fx \rangle \leq \epsilon \epsilon^f \|x\|^2,\]

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

\[\|F^{1/2}\mathbb{1}\|^2 = \langle \mathbb{1}, F\mathbb{1} \rangle = \sum_v e^{\epsilon f(v)} = n \mathbb{E}[e^{\epsilon f(v)}].\]

This establishes the second claim.

Combining the two claims above, we have that \(F^{1/2}RF^{1/2}\) has maximum (absolute) eigenvalue

\[(1 - \epsilon) \mathbb{E}[e^{\epsilon f(x)}] + \epsilon \epsilon^f.\]
Thus $F^{1/2}RF^{1/2}$ has maximum eigenvalue at most

$$
\|F^{1/2}RF^{1/2}\| \leq \epsilon e^\epsilon + (1 - \epsilon) \mathbb{E}[e^{\epsilon f(v)}].
$$

We expand the right hand side by the inequality $e^\epsilon \leq 1 + \epsilon + \epsilon^2$ for small $|t|$, giving the upper bound

$$
\|F^{1/2}RF^{1/2}\| \leq \epsilon \left(1 + \epsilon + \epsilon^2\right) + (1 - \epsilon) \left(1 + \epsilon \mu + \epsilon^2 \mu\right)
$$

$$
= 1 + (1 + t) \epsilon (\mu + \epsilon)
$$

$$
\leq e^{(1+\epsilon)\epsilon(\mu+\epsilon)}.
$$

In turn, for the $k$th power, we have

$$
\left\|\left(F^{1/2}RF^{1/2}\right)^k\right\| = \left\|F^{1/2}RF^{1/2}\right\|^k \leq e^{k(1+\epsilon)\epsilon(\mu+\epsilon)}.
$$

Finally, returning to the original quantity we wanted to sum, we have

$$
\frac{1}{n} \langle F^{1/2}1, F(RF)^k F^{1/2}1 \rangle \leq e^{k(1+\epsilon)\epsilon(\mu+\epsilon)} \mathbb{E}[e^{\epsilon f(v)}] \leq (1 + O(\epsilon)) e^{(k+1+\epsilon)\epsilon(\mu+\epsilon))}.
$$

as desired for Claim 1.

19.5 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

$$
G \mapsto Z \left( G^2 \middle| H \right),
$$

where the degrees and sizes of $G$ and $H$ are appropriately set. The primary 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 $\text{doubly}$ 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 19.6.** Let $H$ be a graph with $d^8$ vertex, degree $d$, and spectral gap $\geq 7/8$. Define graphs $G_1, G_2, \ldots$ by

\begin{align*}
G_1 &= H^2 \\
G_{t+1} &= Z((G_t \otimes G_t)^2 | H)
\end{align*}

Let $n_t$ be the number of vertices in $G_t$. Then $n_t \approx c^{2^t}$, and simulating one step of a random walk in $G_t$ takes $\text{poly}(\log(n_t))$ time.

We first recall the first two lemma’s that we proved previously.

**Lemma 19.7.** 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 $\gamma$, then $R^k$ has spectral gap $1 - (1 - \gamma)^k$.

**Lemma 19.8.** Let $G = (V, E)$ be a regular undirected graph with $n$ vertices and degree $d$, with spectral gap $\gamma_G$. Let $H$ be a regular undirected graph with $d$ vertices and degree $d_0$. Then $Z(G | H)$ is a regular undirected graph with $nd$ vertices, degree $d_0^2$ and spectral gap $\gamma_G^2 \gamma_H^2$.

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 19.9.** 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 $d_1d_2$. Then the random walk matrix of $G_1 \otimes G_2$, denoted $R_1 \otimes R_2 : \mathbb{R}^{V_1 \times V_2} \rightarrow \mathbb{R}^{V_1 \times V_2}$, 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 correspondence between pairs of eigenvectors from $G_1$ to $G_2$, where an eigenvector $v_1$ with eigenvalue $\lambda_1$ of $G_1$ and an eigenvector $v_2$ with eigenvalue $\lambda_2$ of $G_2$ maps to an eigenvector $v_1 \otimes v_2$ of $G_1 \otimes G_2$ with eigenvalue $\lambda_1 \lambda_2$. 

264
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 $G_1, G_2, \ldots$ iteratively by

$$G_{i+1} = \mathcal{Z}\left((G_i \otimes G_i)^2 \big| H\right).$$

The various parameters of interest develop as follows.

| Graph    | $G$ $\rightarrow$ $G \otimes G$ $\rightarrow$ $(G \otimes G)^2$ $\rightarrow$ $\mathcal{Z}\left((G \otimes G)^2 \big| H\right)$ |
|----------|----------------------------------------------------------------------------------------------------------------------------------|
| Vertices | $n$ $\rightarrow$ $n^2$ $\rightarrow$ $n^2$ $\rightarrow$ $n^2d^4$ |
| Degree   | $d^2$ $\rightarrow$ $d^4$ $\rightarrow$ $d^8$ $\rightarrow$ $d^2$ |
| $\gamma$ | $\gamma$ $\rightarrow$ $\gamma$ $\rightarrow$ $2\gamma - \gamma^2$ $\rightarrow$ $(2\gamma - \gamma^2)(7/8)^2$ |

We note that $\gamma \geq 1/2$ implies $(2\gamma - \gamma^2) \geq 1/2$, so the spectral gap never drops below $1/2$.

19.6 Additional notes and references

For more on the topic see [vadhan].
Chapter 20

Randomized Proofs and Verification by Random Walks

20.1 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 member $x \in L$ can be proven in polynomial time. This means there exists a (deterministic) polynomial time algorithm, called the verifier, that takes as input $x$ as well as an additional polynomial sized input $y$ called the proof, and decides if $x \in L$ with the following guarantees. If $x \in L$, then there exists a proof $y$ for which the verifier accepts $x$. If $x \notin L$, then for any proof $y$, the verifier rejects $x$.

Obviously $P \subseteq NP$, and an outstanding open problem is whether $P$ is equal to $NP$. Our experience struggling with 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 beyond computers; see for example [wigderson-09].

It is natural (in hindsight) to consider these questions from a randomized point of view. Suppose we granted the verifier access to randomization, and relaxed our guarantee to have one-sided error. In particular, suppose we relaxed to the following guarantee given $x$ and a proof $y$.

1. If $x \in L$, then the verifier always accepts $x$.
2. If $x \notin L$, then the verifier accepts $x$ with probability $\leq 1/2$.

Of course any language on $NP$ satisfies the above. But the introduction allows us to ask interesting questions.

The proof is accessed in an oracle model, where in one query the verifier can request the $i$th bit from the oracle. For a language $L$, a (nonadaptive) probabilistically checkable proof with $r$ bits and $q$ queries, denoted $PCP(r,q)$ is one that, given an
input $x$ and oracle access to a proof $y$, inspects $x$ and flips at most $O(r)$ random bits. It then makes $O(q)$ queries to the proof. We point out that the verifier can still spend polynomial time inspecting the input $x$ and the outcomes of the $r$ coin flips to choose its queries. Beyond that, the queries to the proof are nonadaptive.

Of course $\text{NP} \leq \text{PCP}(0, \text{poly}(n))$. We are interested

**Theorem 20.1.** *Every NP* language has a probabilistically checkable proof with $O(1)$ random bits and $O(\log n)$ queries to the proof.

The PCP theorem is philosophically very interesting, giving a robust, “error-correcting” extension to our deterministic notion of proofs.\(^1\) The PCP theorem has also had a very big impact in *hardness of approximation*. As we know too well, many problems are NP-hard, and thus we are interested in designing approximation algorithms for them. However it is not known if there are limits to approximations as well; i.e., if we should expect better and better approximations over time for all problems, or if there is a hard limit to the approximation guarantee.

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 $7/8$-th 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 $7/8$-th is optimal:

**Theorem 20.2 ([hastad-2001]).** *For all* $\epsilon > 0$, *it is NP-Hard to obtain a* $(7/8 + \epsilon)$-approximation to 3-SAT.

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 many of 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 that is considered to be simpler then the original proof of the PCP theorem. In particular, the machinery driving Dinur’s approach will hopefully be somewhat familiar based on our many discussions already on random walks.

### 20.2 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 $v_1, \ldots, v_k \in V$ and a subset $S \subseteq A^k$. An assignment $\sigma : V \rightarrow A$ satisfies

\(^1\)The old joke is that the PCP theorem implies a much faster way to grade algorithms homework.
the constraint if \((\sigma(v_1), \ldots, \sigma(v_k)) \in S\). 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.

Our discussion is about proving the above theorem in particular.

**Theorem 20.3.** There are integers \(k > 1, |A| > 1\) such that, given a collection of \(k\)-ary constraints over an alphabet \(\Sigma\), it is NP-hard to decide whether

1. All clauses can be satisfied.
2. Less than or equal to half the clauses can be satisfied.

It is equivalent to the PCP theorem.

**Theorem 20.4.** Theorems 20.1 and 20.3 are equivalent.

We leave the proof as the following exercise.

**Exercise 20.1.** Prove theorem 20.1 and theorem 20.3 are equivalent. Below we give part of the proofs, in both directions, to get you started.

1. **Theorem 20.1 \(\Rightarrow\) theorem 20.3.** Suppose the PCP theorem, theorem 20.1, 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 queries \(q = O(1)\) 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 \(\text{unsat}(P) \geq 1/2\). 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 20.3 \(\Rightarrow\) theorem 20.1.** 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 to 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 \(\text{unsat}(P_x) = 0\) and \(x \notin L\) iff \(\text{unsat}(P_x) \geq 1/2\). We create a probabilistically checkable proof system where...
20.3 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 20.3.

Now, binary CSP is easier to visualize because we can imagine the clauses arranged in a graph. 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 - which will be advantageous when we start modifying the problem - we will 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 $\text{unsat}(G) = 0$. Since 3-colorability has one constraint per edge we can recast this as saying that it is NP-Hard to decide if $\text{unsat}(G) = 0$ or if $\text{unsat}(G) \geq 1/m$, 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 that it is NP-Hard to distinguish between $\text{unsat}(G) = 0$ or $\text{unsat}(G) \geq c$, for any fixed constant $c$.

20.3.1 Iterative amplification - an overview

For inspiration, we briefly recall the deterministic logspace algorithm for $(s,t)$-connectivity [reingold]. We knew that this problem was actually easy for constant degree expanders, but of course the input graph is not in general 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. Applying these operations together gave 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 is very much similar in spirit, trying to amplify a graph CSP by various transformations and gradually. In fact dinur cites the $(s,t)$-connectivity algorithm [reingold] as an inspiration.

\(^2\)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.
1. *Expander-ification*, where we guarantee that the graph CSP is a regular graph with constant degree and has constant expansion gap.

2. *Error amplification by powering*, where we amplify the UNSAT of the graph by taking a power of the graph and creating constraints appropriately.

3. *Alphabet reduction*. Where we reduce the alphabet size of the graph CSP (which increases in the powering stage).

Each of these three steps take as input one graph CSP $G$ and output another, $G'$. As we analyze these steps, we will carefully pay attention to the following properties in particular which are essential to maintaining the high-level correctness.

1. **Completeness**: If $\text{UNSAT}(G) = 0$, then $\text{UNSAT}(G') = 0$.

2. **Soundness**: $\text{UNSAT}(G') \geq \beta \text{UNSAT}(G)$ for some value $\beta > 0$.

Ultimately we want $\beta > 1$, but the operations below may also have $\beta < 0$ as they are principally concerned with managing other parameters (such as the alphabet size, or the degree).

However in one round trip we are able to show that we have a net gain of $\beta > 1$, for a fixed constant $\beta$, while keeping the other parameters under control (and more precisely, universal constants).

Let us break down these steps in a little more detail. We will specify the context and state the key lemma’s.

### Degree reduction

**Lemma 20.5.** There exists constants $d \in \mathbb{N}$, $\gamma > 0$, and $\beta > 0$ for which the following hold. 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. $\beta_1 \text{UNSAT}(G) \leq \text{UNSAT}(G') \leq \text{UNSAT}(G)$ for some universal constant $\beta_1$.

4. The spectral gap on $G'$ is $\geq \gamma$.

5. The size of $G'$ is at most a constant factor bigger than the size of $G$. 


**Error Amplification**

**Lemma 20.6.** 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 whose random walk has spectral gap (at least) $\gamma$. Then the $t$-th power $G^t$ (as described above) has the following properties.

1. $G^t$ is regular with degree $d^t$, alphabet $A^{d^t}$, and the random walk on $G^t$ has spectral gap $1 - (1 - \gamma)^t$.

2. If $\text{unsat}(G) = 0$, then $\text{unsat}(G^t) = 0$.

3. $\text{unsat}(G^t) \geq \beta_2 \sqrt{t} \min\{\text{unsat}(G), \frac{1}{t}\}$.

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 $G$ in the parameter $\beta_2$.

**Alphabet reduction**

**Lemma 20.7.** 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

$$\beta_3 \text{ unsat}(G) \leq \text{unsat}(G') \leq \text{unsat}(G).$$

### 20.3.2 Putting it all together

<table>
<thead>
<tr>
<th>Lemma</th>
<th>20.5</th>
<th>20.6</th>
<th>20.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree</td>
<td>$d_0$</td>
<td>$d_0^t$</td>
<td>$d_0^t$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$\gamma_0$</td>
<td>$1 - (1 - \gamma)^t$</td>
<td>$1 - (1 - \gamma)^t$</td>
</tr>
<tr>
<td>$\text{unsat}(G)$</td>
<td>$\alpha$</td>
<td>$\beta_1 \alpha$</td>
<td>$\sqrt{t} \beta_2 \beta_1 \alpha$</td>
</tr>
<tr>
<td>$A_0$</td>
<td>$A_0$</td>
<td>$A_0^{\left\lfloor \frac{t}{2} \right\rfloor}$</td>
<td>$A_0$</td>
</tr>
</tbody>
</table>

In the row for $\text{unsat}(G)$, we assume that $\beta_1 \alpha \leq \frac{1}{t}$, since otherwise $\text{unsat}(G)$ is at least a constant and we are done.
20. Randomized Proofs and Verification by Random Walks

20.4. Expander-ification

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 in the expander, we assign the “equality constraint” \( C_e = \{(a, a) : a \in A\} \subset A^2 \).

This type of construction first appears in [py].

**Lemma 20.8.** Given a graph CSP \( G \), let \( G' \) be the graph CSP obtained by the construction described above. Then:

1. The total number of edges of \( G' \) is within a constant factor of the number of edges of \( G \).
2. \( G' \) has the same alphabet as \( G \).
3. \( G' \) is a \((d_0 + 1)\)-regular graph for a universal constant \( d_0 \).

Figure 20.1: Reducing the degree of a graph CSP by splitting each vertex into a constant degree expander of equality constraints.
20. Randomized Proofs and Verification by Random Walks

20.5 Error amplification

We now turn to the error amplification stage. Let us briefly recall the context. We have a graph CSP $G$, which is either satisfiable or has some nonzero $\text{unsat}(G)$. The goal is to produce a new constraint CSP $G'$ that (a) is satisfiable iff $G$ is satisfiable, and (b) has substantially larger $\text{unsat}(G')$ 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 offers the following helpful intuition. Fix an assignment $\pi : V \rightarrow A$, and suppose we sample $t$ edges in $G$ uniformly at random and see if any of the corresponding constraints are not satisfied. The probability that at least one of them is unsatisfied is $1 - (1 - \text{unsat}(G))^t \approx t \cdot \text{unsat}(G)$ (for $\text{unsat}(G)$ 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 $\text{unsat}$ value $1 - (1 - \text{unsat}(G))^t \approx t \cdot \text{unsat}(G)$, which is good. Some drawbacks of this construction are that (a) the resulting CSP is no longer binary, and (b) the number of constraints in the CSP increases substantially from $m$ to $\approx m^t$.

To summarize, we can increase $\text{unsat}$ by independent repetition but encoding this directly as a 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 is analytically similarly to an independent sample. To take advantage of this, we try to create a CSP instance on top of the $t$-th power of $G$ as a graph.

We briefly described the construction in ?? but let us describe it anew. Consider the graph $G^t$ that (informally speaking) 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 $(v_0, v_1)$ in $G_t$ for every $t$-step walk $(v_0, \ldots, v_t)$ in $G_L$. We use the alphabet $A^P$ for $D = \sum_{i=0}^{t} d^i = O(d^t)$. Note that for every vertex $v$, there are at most $D$ vertices within $t$ steps of $v$ in $G$. We identify an $A^P$-label $\bar{\pi}(v)$ as an $A$-labeling of all the vertices withing $t$-steps of $v$, letting $\bar{\pi}(v, w) \in A$ denote the label assigned by $\bar{\pi}(v)$ to $w$. For each $t$-step walk $w = (v_0, \ldots, v_t)$ in $G_L$, which corresponds to
20. Randomized Proofs and Verification by Random Walks

20.5. Error amplification

For each edge \(e = (v_i, v_{i+1}) \in w\), \((\bar{\pi}(v_0), \bar{\pi}(v_i), \bar{\pi}(v_{i+1})) \in A^2\) satisfies the constraint \(C_e\) in \(G\).

It is easy to see that a satisfying assignment for \(G\) gives a satisfying assignment for \(G'\). In the converse direction, we want to show that if \(G\) has nonzero \(\text{unsat}(G)\), then \(G'\) has substantially larger \(\text{unsat}(G)\). In particular we will show that \(\text{unsat}(G') \geq \text{poly}(t) \text{unsat}(G)\). We prove this via the contrapositive: given any labeling \(\bar{\pi} : V \rightarrow A^D\), we derive a labeling \(\pi : V \rightarrow A\) and show that \(\text{unsat}(\pi | G) \leq \text{poly}(1/t) \text{unsat}(\bar{\pi} | G')\). The labeling \(\pi\) is derived as follows. Fix a vertex \(v_0 \in V\). To choose \(\pi(v_0)\), consider a lazy \((t/2)\)-step random walk \(v_0, \ldots, v_{t/2}\), and the \(A\)-label \(\bar{\pi}(v_{t/2}, v_0)\) for \(v_0\) induced by the \(A\)-label at the destination \(v_{t/2}\). We choose \(\pi(v_0)\) to be the most likely \(A\)-label to be assigned to \(v_0\). Note that \(\mathbb{P}[\bar{\pi}(v_{t/2}, v_0) = \pi(v_0)] \geq 1/|A|\), which for the sake of our conversation, is at least some constant.

\textbf{dinur} offers the following interpretation for \(\pi\). For each vertex \(v\), \(\bar{\pi}(v)\) assigns \(A\)-labels to all vertices close to \(v\). For each vertex \(x\), \(\pi(x)\) takes the most popular opinion among its neighbors. Here “popularity” is weighted according to random walks from \(x\). Let us informally suggest that if we had taken the popular opinion over lazy \(i\)-step walks, for \(i\) very close to \(t/2\) (such as \(t/2 \pm O\left(\sqrt{t}\right)\)), we would have similar labelings because the lengths of the walks are so similar (this will be formalized shortly).

The main effort in this section is to show that \(\text{unsat}(\pi | G) \leq \text{unsat}(\bar{\pi} | G')/O\left(\sqrt{t}\right)\) (assuming \(\text{unsat}(\pi | G)\) is not already at least \(1/t\)). Why should this be the case? Speaking informally, let \(C_e\) be a constraint failed by \(\pi\), where \(e = (u, v)\). Then the “popular opinion” about \(u\) and \(v\) fail constraint \(C_e\). To relate this to \(t\)-step walks, consider the \(t\)-step lazy walks \((v_0, \ldots, v_t)\) where \(e\) appears close to the middle; say, within the middle \(\sqrt{t}\) edges. Since the length from the endpoints of \(e\) to \(v_0\) and \(v_t\) are roughly \(t/2\), the distribution of labels \(\pi(v_0 | u)\) and \(\pi(v_t | v)\) will be similar to the distribution over \(t/2\)-step walks used to generate \(\pi\). In particular, \(\pi(v_0 | u)\) and \(\pi(v_t | v)\) should have a constant fraction change of agreeing with \(\pi(u)\) and \(\pi(v)\). That is, a random walk \(w\) with \(e\) amongst the middle \(O\left(\sqrt{t}\right)\) edges is an unsatisfied constraint in \(G'\) with constant probability.

Thus every unsatisfied edge \(e\) in \(G\) implies about \(\sqrt{t}\) unsatisfied edges in \(G'\). But this does not immediately imply that \(\text{unsat}(\bar{\pi} | G')\) is a \(\Omega\left(\sqrt{t}\right)\)-factor larger than
unsat(π | G), because we are counting the number of failed edges in G′ with repetition. Maybe all of the unsatisfied edges e imply failures for the same set of unsatisfied edges in G′. Here enters the fact that G is an expander. If we take a random walk in G, and this random walk behaves like a uniformly random sample, then whether or not one edge is failed by π is not strongly correlated with whether or not subsequent edges are also failed by π. Consequently the set of bad walks for ¯π corresponding to a bad edge e for π is relatively independent across e.

To start formalizing this intuition, let I = \{i : |i − t/2| ≤ \sqrt{t/2}\}. This interval represents roughly a standard deviation around the mean for the sum of t/2 independent coin tosses. In particular the likelihood of i/2 heads out of t/2 coin tosses is similar for all i ∈ I. For the same reasons, the distribution of the number of heads out of i coin tosses is similar for all i ∈ T. While π are derived specifically from (t/2)-step lazy random walks, probabilistically speaking, the labels would have been similar for an i-step walk for any i ∈ I, as follows.

Lemma 20.9. For any fixed |A|, there exists a value T ∈ \N and a constant c > 0 such that the following holds for all t ≥ T.

Let v0 ∈ V, and let v0, v1, v2, ... by a lazy random walk from v0 in G. Then for all i with |i − t/2| ≤ \sqrt{t/2},

P[¯π(vi, v0) = π(v0)] ≥ c \cdot P[¯π(v_{i/2}, v_0) = π(v_0)] ≥ c/|A|.

Let us briefly sketch the intuition but postpone the proof to the end of the section. The key idea is that the random walk is lazy, staying put on any particular vertex with probability 1/2. An alternative way to generate a lazy walk of i-steps is to first flip coins for each step and decide how many of the i-steps will be non-lazy. Call this number j. Then we take a (non-lazy) walk in G for j steps. The distribution of ¯π(vi, v0), conditional on j, is independent of i. The claim is that the distribution for j is roughly the same for all i ∈ I. Intuitively, j is concentrated around i/2 ± O(\sqrt{i}). But since i/2 ≈ t/2 for all i ∈ I, j is also concentrated around t/2 for all i ∈ I. That is to say that j has a similar distribution regardless of the choice of j ∈ I.

Now, recall the high level intuition is that the edges along random walks should behave almost independently when G is an expander. For example, fix any set F ⊂ E, and let μ = |F|/|E|. (Think of F is as the edges failed by π.) Consider a random walk in G of which some of the edges are from F and others are not. If the random walk behaved like an independently random sample of edges, then conditional on whether or not the most recent step of the walk was in F, the next step would be in F with probability μ. The following lemma shows this holds approximately for expander graphs.
Lemma 20.10. 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 random edge from $F$. Then

$$\Pr[(v_i, v_{i+1}) \in F] \leq \mu + (1 - \gamma)^i.$$ 

Let us postpone the proof of this lemma as well until the end of the section. We now turn to the main claim of this section. In the following, note that we will take $t$ to be a constant, so if the minimum is attained by $1/t$, then $\text{unsat}(\pi \mid G)$ is a constant and we are done.

Lemma 20.11. $\text{unsat}(\bar{\pi} \mid G') \geq \Omega\left(\sqrt{t}\right) \min\left\{ \text{unsat}(\pi \mid G), \frac{1}{t} \right\}$. 

Proof. Let $F \subset E$ be the set of edges failed by $\pi$, and let $\mu = |F|/|E|$. If $\mu \geq 1/t$, then drop edges from $F$ until $\mu \leq 1/t$. We have $1/t \geq \mu \geq \Omega(1) \min\left\{ \text{unsat}(\pi \mid G), \frac{1}{t} \right\}$.

For $i \in I$, we create an random indicator variable $X_i \in \{0, 1\}$ as follows. 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 both

1. The $i$th edge in $w$, $e_i = (v_{i-1}, v_i)$, has labels $\bar{\pi}(v_{i-1}, v_0) = 1$ and $\bar{\pi}(v_i, v_t) = 1$.
2. $\pi$ fails to satisfy the $i$th edge in $w$, $e_i$.

Let $X = \sum_{i \in I} X_i$. Observe that $\text{unsat}(G') \geq \Pr[X > 0]$. We make two claims about $X$.

Claim 20.5.1. For all $i \in I$, $\E[X_i] \geq \Omega(1)\mu$.

Observe that by linearity of expectation, claim 20.5.1 implies that $\E[X] \geq \Omega\left(\sqrt{t}\right)\mu$.

Claim 20.5.2. $\E[X^2] \leq O\left(\sqrt{t}\right)\mu$.

We will prove the claims later. Let us first assume they hold and complete the proof. We have

$$\E[X^2] = \E[X^2 \mid X > 0] \Pr[X > 0] \geq \E[X \mid X > 0]^2 \Pr[X > 0] = \frac{\E[X]^2}{\Pr[X > 0]}$$

where (a) is by Jansen’s inequality (w/r/t the convex function $f(x) = x^2$). Rearranging and applying the claims, we have

$$\Pr[X > 0] = \frac{\E[X^2]}{\E[X]^2} \geq \Omega\left(\sqrt{t}\right)\mu,$$

as desired. \qed

276
It remains to prove the claims, starting with claim 20.5.1. We restate the claim for the reader’s convenience.

**Claim 20.5.1.** For all \( i \in I \), \( \mathbf{E}[X_i] \geq \Omega(1)\mu \).

**Proof.** 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 take the walk \( w = (v_0, \ldots, v_t) \). This produces a uniformly random lazy walk because \( G \) is regular. We have

\[
P[X_i = 1] = \mu \cdot P[\tilde{\pi}(v_0, v_{i-1}) = \pi(v_{i-1})] \cdot P[\tilde{\pi}(v_t, v_i) = \pi(v_i)].
\]

Recall from lemma 20.9 that, for \( i \in I \), the marginals of \( \tilde{\pi}(v_t, v_i) \) are within a constant factor of \( \pi(v_t, v_{t/2}) \), which in turn is at least \( 1/|A| \). Similarly for \( \tilde{\pi}(v_0, v_i) \). Thus

\[
P[X_i = 1] \geq \Omega(1)\mu/|A|^2,
\]

as desired. \( \square \)

The remaining claim, claim 20.5.1, is about the variance of \( X = \sum_{i \in I} X_i \). To this end we have an intermediate claim analyzing the cross-terms \( X_iX_j \).

**Claim 20.5.3.** Let \( i, j \in I \) with \( i \neq j \). Then \( \mathbf{E}[X_iX_j] \leq \mu \left( \mu + (1 - \gamma)^{j-i-1} \right) \).

**Proof.** Let us define random indicators variables \( Y_i, Y_j \in \{0, 1\} \) that indicate whether the \( i \)th edge is in \( F \). Then \( 0 \leq X_i \leq Y_i \) and so \( \mathbf{E}[X_iX_j] \leq \mathbf{E}[Y_iY_j] \). Write

\[
\mathbf{E}[Y_iY_j] = \mathbf{E}[Y_i] \cdot \mathbf{E}[Y_j | Y_i = 1] = \mu \cdot \mathbf{E}[Y_j | Y_i = 1].
\]

The remaining term, \( \mathbf{E}[Y_j | Y_i = 1] \), 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 20.10, this probability is at most \( \mu + (1 - \gamma)^{j-i-1} \). \( \square \)

Now we prove claim 20.5.2, restating it first for the reader’s convenience.

**Claim 20.5.2.** \( \mathbf{E}[X^2] \leq O(\sqrt{t})\mu \).

**Proof.** We have

\[
\mathbf{E}[X^2] = \sum_{i \in I} \mathbf{E}[X_i^2] + 2 \sum_{i < j} \mathbf{E}[X_iX_j]
\]

\[
= \mu|I| + 2\mu^2|I|^2 + 2\mu \sum_{i,j \in I} (1 - \gamma)^{j-i-1}
\]

\[
\overset{(a)}{=} O(1)\mu|I| + 2\mu^2|I|^2
\]

\[
\geq O(\sqrt{t})\mu.
\]
Here (a) is by claim 20.5.3. (b) is because $\mu \geq 1/t$ and $|I| = O\left(\sqrt{t}\right)$. \qed

This establishes, modulo lemma 20.9 and lemma 20.10 which were introduced earlier in the section. The remainder of this section is devoted to proving these lemma’s.

### 20.5.1 An expander mixing lemma for edges

Let us first prove lemma 20.10 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 20.5.2 will fail).

**Lemma 20.10.** 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

$$P[(v_i, v_{i+1}) \in F] \leq \mu + (1 - \gamma)^i.$$  

**Proof.** 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 $t$th step is in $F$ is exactly

$$P[(v_{t-1}, v_t) \in F] = \langle y, R^{t-1}x \rangle = \frac{2|F|}{d} \langle x, R^{t-1}x \rangle.$$  

Since $R$ is a regular undirected graph with spectral gap $\gamma$, we can write

$$R^{t-1} = \frac{1}{\gamma} (1 \otimes 1) + R'$$  

where $\|R'\| \leq \gamma^{t-1}$. The remainder of the proof is reserved for the following exercise. \qed
Exercise 20.3. Complete the remainder of the proof by the following steps.

1. Show that $\langle x, R^{t-1}x \rangle \leq \frac{1}{n} + \frac{\gamma t - 1}{d |F|}$.

2. Show that $P[(v_{t-1}, v_t) \in F] \leq \frac{|F|}{|E|} + \gamma^{t-1}$.

20.5.2 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 20.12. 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

$$z \leq P[B_n = k] \leq \frac{1}{z}$$

The proof of lemma 20.12 is given as ???. Let us continue to prove lemma 20.9.

Lemma 20.9. For any fixed $|A|$, there exists a value $T \in \mathbb{N}$ and a constant $c > 0$ such that the following holds for all $t \geq T$.

Let $v_0 \in V$, and let $v_0, v_1, v_2, \ldots$ by a lazy random walk from $v_0$ in $G$. Then for all $i$ with $|i - t/2| \leq \sqrt{t/2}$,

$$P[\overline{\pi}(v_i, v_0) = \pi(v_0)] \geq c \cdot P[\overline{\pi}(v_{t/2}, v_0) = \pi(v_0)] \geq c/|A|.$$ 

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 $|B_{t/2} - t/4| \geq c \sqrt{t} \leq 1/2|A|$. For ease of notation, let $E_i$ be the event that $\pi(v_i | v_0) = \pi(v_0)$. We want to show that for $i \in I$, we have $P[E_i] \geq \Omega(1) P[|E_{t/2}|]$. We have

$$P[E_i] \geq \sum_{j:|j-t/4|\leq c\sqrt{t/2}} P[E_i | B_i = j] P[B_i = j] \geq \sum_{j:|j-t/4|\leq c\sqrt{t/2}} z P[E_i | B_i = j] P[B_{t/2} = j] \geq \sum_{j:|j-t/4|\leq c\sqrt{t/2}} P[E_{t/2} | B_{t/2} = j] P[B_{t/2} = j] \geq z \left( P[E_{t/2}] - \frac{1}{2|A|} \right) \geq \frac{z}{2} P[E_{t/2}].$$
20. Randomized Proofs and Verification by Random Walks

20.6 Alphabet reduction

This section is about the third graph-CSP transformation reduction, where goal the set of satisfying assignments for the constraint labeled by boolean formulas over the endpoints defining a constraint.

Figure 20.2: A high-level schematic for the alphabet reducing transformation in section 20.6, on a triangle graph.

as desired. Here (a) is by lemma 20.12, for $t$ sufficiently large and $z$ the constant asserted in lemma 20.12. The reason for (b) is left as an exercise below. (c) is by choice of $c$. (d) is because $P[E_{t/2}] \geq 1/|A|$.

Exercise 20.4. Justify equality (b) in the proof of lemma 20.9.

20.6 Alphabet reduction

This section is about the third graph-CSP transformation reduction, where goal 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 $C : A \to \{0, 1\}^\ell$ 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 $C(a_1), C(a_2) \in \{0, 1\}^\ell$ differ in at least $\rho \ell$ bits. We will create a uniform $(2\ell)$-ary CSP with binary alphabet $\{0, 1\}$. Such a CSP can be interpreted as a $(2\ell)$-uniform hypergraph with hyperedges labeled by boolean formulas over the endpoints defining a constraint.

For each vertex $u$, let $V_u = \{u_1, \ldots, u_\ell\}$ be a new set of $\ell$ 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

$$f_e(x_u, x_v) = \begin{cases} 1 & \text{if } x_u = C(a_1) \text{ and } x_v = C(a_2) \text{ for some } (a_1, a_2) \in C_e, \\ 0 & \text{otherwise;} \end{cases}$$

where we identify a boolean assignment $x_u : V_u \to \{0, 1\}$ as a length $\ell$ bit string $(x_u(u_1), \ldots, x_u(u_\ell))$. 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 $f_e$.  

280
Step 2: Replace each hyperedge constraint with a graph CSP. We want to convert the hypergraph generated above into a graph. We will apply a certain operation hyperedge-wise that reduces each hyperedge independently to a graph CSP. For the time being we will treat this as a black box and address how to do this separately later. Let $A_0$ be a finite alphabet and $\epsilon_0 > 0$ be parameters 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 $V_u, V_v \subseteq V_e$ with the following properties.

1. If $f_e(x_u, x_v) = 1$, then we can extend $(x_u, x_v) : V_u \cup V_v \rightarrow \{0, 1\}$ to an assignment $\sigma : V_e \rightarrow A_0$ that satisfies all of $G_e$. This is called the completeness property.

2. If $f_e(x_u, x_v) = 0$, and the relative distance from $(x_u, x_v)$ to $F_e$ is at least $\delta$ then any extension $\sigma : V_e \rightarrow A_0$ of $(x_u, x_v)$ has error at least $\epsilon_0 \delta$ for some fixed $\epsilon_0$. That is,

$$\text{UNSAT}(G_e) \geq \epsilon_0 \text{dist}((x_u, x_v), F_e),$$

where $\text{dist}(a, b)$ denotes the relative Hamming distance between $a$ and $b$. This is called the soundness property.

Moreover, one prove that for some parameters $A_0$ and $\epsilon_0$ (depending only on $|\ell|)$, we can create the above instances for $G_e$ such that $|E_e|$ is the same across all $e$.

Since we will not prove how to do this step just yet, let us instead provide some comments. For each edge $e$, we transform a boolean function on $2\ell$ variables to a graph CSP with finite alphabet size and possibly a few more variables. There was 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 $\tilde{O}(2^\ell)$ bits, and $\ell$ is independent of $n$. So however much things may blow up, 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.

20.6.1 Analysis

Thus we have a two-step process that takes one graphical CSP and produces another with a constant alphabet size, which is intermediate also a hypergraph CSP over the alphabet $\{0, 1\}$. We assume for the time being that step 2 is implementable, and (might) address that aspect later. (Otherwise, see [dinur], or [odonnell])

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 20.13. \( \text{unsat}(\bar{G}) \leq \text{unsat}(G) \).

Proof. Let \( \pi : V \to A \) be an assignment that attains \( \text{unsat}(G) \). 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 \( C(\pi(u)) \).
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 \( \pi \), 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 \( \text{unsat}(\bar{G}) \leq \text{unsat}(G) \).

\( \square \)

Lemma 20.14. For \( \beta_3 = \epsilon_0 \rho/4 \), we have
\[
\beta_3 \text{unsat}(G) \leq \text{unsat}(\bar{G}).
\]

Proof. Let \( \bar{\pi} : \bar{V} \to A_0 \) be an assignment for \( \bar{\pi} \) that attains \( \text{unsat}(\bar{G}) \). 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 \( \pi \), a \( \beta_3 \)-fraction of \( G_e \) for some value \( \beta \) that depends on \( |A_0| \) and \( \rho \). It then follows that \( \text{unsat}(\bar{G}) \geq \beta_3 \text{unsat}(G) \).

Let \( e = (u, v) \) be an edge that is not satisfied by \( \pi \). For ease of notation, let us denote
\[
\bar{u} \overset{\text{def}}{=} (\bar{u}_1, \ldots, \bar{u}_\ell) \text{ and } \bar{\pi}(\bar{u}) \overset{\text{def}}{=} (\bar{\pi}(u_1), \ldots, \bar{\pi}(u_\ell)).
\]
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 \( \rho \) 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 \( G_e \). \( \square \)
20. Randomized Proofs and Verification by Random Walks

20.7 Exercises

Exercise 20.1. Prove theorem 20.1 and theorem 20.3 are equivalent. Below we give part of the proofs, in both directions, to get you started.

1. **Theorem 20.1 \(\implies\) theorem 20.3.** Suppose the PCP theorem, theorem 20.1, 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 queries \(q = O(1)\) 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 \(\text{unsat}(P) \geq 1/2\) for a given CSP problem \(P\). This means to 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 \(\text{unsat}(P_x) = 0\) and \(x \notin L\) iff \(\text{unsat}(P_x) \geq 1/2\). We create a probabilisticaly checkable proof system where...

2. **Theorem 20.3 \(\implies\) theorem 20.1.** 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 to for every language \(L\), 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 ...

Exercise 20.2. 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

\[
z \leq \frac{P[B_n = k]}{P[B_m = k]} \leq \frac{1}{z}
\]

Exercise 20.3. Complete the remainder of the proof by the following steps.

1. Show that \(\langle x, R^{t-1}x \rangle \leq \frac{1}{n} + \frac{\gamma^{t-1}d}{2|F|}\).
2. Show that $P[(v_{t-1}, v_t) \in F] \leq \frac{|F|}{|E|} + \gamma^{t-1}$.

**Exercise 20.4.** Justify equality (b) in the proof of lemma 20.9.
Chapter 21

Randomly Testing Boolean Functions

21.1 Testing boolean formulae with 3 queries, and 8 letter graph CSP’s

A boolean function is a function \( f : \{0, 1\}^n \rightarrow \mathbb{R} \) 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 \( f \). However, the same universality of boolean functions that makes them so attractive also makes it seem rather daunting to be able to obtain concrete and useful observations. 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 \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) satisfies \( f(x + y) = f(x) + f(y) \) for all \( x \) and \( y \).
- Dictatorship: whether \( f : \{0, 1\}^n \rightarrow \{-1, 1\} \) is of the form \( f(x) = (-1)^{x_i} \) for some \( i \in [n] \).

The main goal of today’s discussion is to describe the following universal tester for proof systems.

\(^1\)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 \( f : \{0, 1\}^n \rightarrow \{-1, 1\} \).
Theorem 21.1. Let $L \subseteq \{0,1\}^n$. Let $p = 2^n$. Then there is a randomized algorithm $A_L : \{0,1\}^n \times \{0,1\}^p \rightarrow \{0,1\}$ that, given oracle access to $x \in \{0,1\}^n$ and $y \in \{0,1\}^p$, has the following properties.

1. $A_L$ makes 3 queries to bits of $x$ or $y$.
2. If $x \in L$, then there exists $y \in \{0,1\}^p$ such that $A_L(x,y) = 1$ always.
3. If $x \notin L$ then for all $y \in \{0,1\}^p$, we have

$$\Pr[A_L(x,y) = 0] \geq .001 \min_{y \in L} \frac{\|x - y\|_0}{n}.$$  

The connection to boolean functions is not at all clear from the theorem statement above. Let us briefly describe the algorithm underlying theorem 21.1 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 $\chi_i : \{0,1\}^N \rightarrow \{-1,1\}$ be the function defined by

$$\chi_i(x) = \begin{cases} 1 & \text{if } x_0 = 0 \\ -1 & \text{if } x_i = 1 \end{cases}$$

$\chi_i$ is called a dictator function and is the topic of section 21.4. 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 $f : \{0,1\}^N \rightarrow \{-1,1\}$ can be expressed in $(2^N = 2^{2^n})$-dimensional vectors. We will create a test that, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^p$, simultaneously tests if $y \in D_L$ and, 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 return to this analysis at the end.

To motivate theorem 21.1, recall that an important ingredient of the PCP theorem from previous discussions was a subroutine that took constant-size boolean functions and output graph CSP’s that model them in an error preserving fashion. Let us now show how to obtain this result using the universal tester above.

Theorem 21.2. Let $L \subset \{0,1\}^n$ be a language. Then there exists 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. $A = \{0,1,\ldots,7\}$.  

2. There are $n$ vertices $X_1, \ldots, X_n \in V_L$ that only take labels in $\{0,1\} \subset A$, and satisfy the following.

- If $x_1, \ldots, x_n \in \{0,1\}$ is such that $(x_1, \ldots, x_n) \in L$, then there exists an assignment $\sigma : V_L \to \{0,1\}$ such that $\sigma(X_i) = x_i$ for all $i$ and $\text{unsat}(\sigma \mid G_L) = 0$.
- If $x_1, \ldots, x_n \in \{0,1\}$ is such that $(x_1, \ldots, x_n) \notin L$, then for all assignments $\sigma : V_L \to \{0,1\}$ such that $\sigma(X_i) = x_i$ for all $i$, we have

$$\text{unsat}(\sigma \mid G_L) \geq .001 \min_{y \in L} \frac{\|x - y\|_0}{n}.$$ 

**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 $p = 2^a$. 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 $\omega$, we create a vertex $Z_\omega$.

We have an alphabet $A = \{0, \ldots, 7\}$ which we identify with $\{0,1\}^3$. For every outcome of coin tosses $\omega$, we create 4 constraints/edges involving $Z_\omega$ based on the mechanism of the tester $T$ when the coin tosses are $\omega$.

1. We create a self-loop at $Z_\omega$ 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 $\omega$, 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 $i$th query of $T$ is to the $k_i$th bit of $x$. Then we create a constraint between $Z_\omega$ and $X_{k_i}$ that accepts iff $\sigma_i(Z_\omega) = \sigma(X_{k_i})$. Similarly, if instead the $i$th query is to $k_i$th bit of $y$, we make the same constraint between $Z_\omega$ 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 $L$. 

\hfill $\Box$
While this note 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 [odonnell] for a booklength treatment on these topics. These notes are based on chapters 1, 2, and 7 of [odonnell]. Property testing also extends far beyond boolean functions. We recommend Prof. Grigorescu’s Spring 2021 class on sublinear time algorithms for more topics in this area.

21.2 Fourier analysis of boolean functions

A boolean function is a real-valued function defined over bit-strings of a fixed length; i.e.,

\[ f : \{0, 1\}^n \rightarrow \mathbb{R}. \]

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 \(i\)th 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 \(\langle \cdot, \cdot \rangle_b\):

\[
\langle f, g \rangle_b \overset{\text{def}}{=} \frac{1}{2^n} \langle f, g \rangle = \mathbb{E}_{x \sim \{0,1\}^n} [f(x)g(x)],
\]

where in the RHS \(x\) is sampled uniformly from \(\{0, 1\}^n\). Let

\[
\|f\|_b = \sqrt{\langle f, f \rangle_b} = \sqrt{\mathbb{E}_x[f^2(x)]}
\]

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 21.3.** For two boolean functions \(f, g : \{0, 1\}^n \rightarrow \{0, 1\}\), we have

\[
\|f - g\|_b^2 = \mathbb{P}_x[f(x) \neq g(x)].
\]

**Lemma 21.4.** For two boolean functions \(f, g : \{0, 1\}^n \rightarrow \{-1, 1\}\), we have

\[
\mathbb{P}[f(x) \neq g(x)] = \frac{1}{4}\|f - g\|_b^2.
\]

**Lemma 21.5.** For any boolean function \(f : \{0, 1\}^n \rightarrow \{-1, 1\}\), we have \(\|f\|_b^2 = 1\).
21. Randomly Testing Boolean Functions

21.2. Fourier analysis of boolean functions

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 \(\chi_S : \{0,1\}^n \rightarrow \{-1,1\}\) by

\[
\chi_S(x) = (-1)^{\sum_{i \in S} x_i}
\]

where \(S \Delta T = (S \cup T) \setminus (S \cap T)\) denotes the symmetric difference. We also have, for all nonempty sets \(S \neq \emptyset\),

\[
E[\chi_S(x)] = 0.
\]

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

\[
E[\chi_S] \overset{(a)}{=} E[\chi_{S'}(x)\chi_i(x)] \overset{(b)}{=} E[\chi_{S'}(x)]E[\chi_i(x)] \overset{(c)}{=} 0.
\]

Here (a) is by (21.1). (b) is by independence. (c) applies the singleton case. Finally, by combining the above observations, we have

\[
\langle \chi_S, \chi_T \rangle_b = \begin{cases} 1 & \text{if } S = T \\ 0 & \text{otherwise} \end{cases}
\]

for any two sets \(S, T \subset [n]\).

The last identity signifies that the set of functions \(\{\chi_S : S \subseteq [n]\}\) is 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 orthonormal basis \((w/r/t \langle \cdot, \cdot \rangle_b)\). Linear algebra then dictates that any boolean function \(f : \{0,1\}^n \rightarrow \mathbb{R}\) can be written uniquely as a linear combination of the Fourier basis functions \(\{\chi_S : S \subseteq [n]\}\), and this representation is given by

\[
f = \sum_{S \subseteq [n]} \langle f, \chi_S \rangle_b \chi_S = \sum_{S \subseteq [n]} E_x[f(x)\chi_S(x)]\chi_S.
\]
Let \( \hat{f} : 2^n \to \mathbb{R} \) denote the coordinates in this basis; i.e., \( \hat{f}_S = \langle f, \chi_S \rangle_b \) for each set \( S \subseteq [n] \). The map \( f \mapsto \hat{f} \) is unitary; that is, a rotation that preserves distances. Consequently for boolean functions \( f, g : \{0,1\}^n \to \{0,1\} \) we have

\[
P_x[f(x) \neq g(x)] = \langle f - g, f - g \rangle_b = \langle \hat{f} - \hat{g}, \hat{f} - \hat{g} \rangle = \|\hat{f} - \hat{g}\|^2,
\]

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.

### 21.3 Linearity

#### 21.3.1 Testing 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 \(-1\), there is an easy 1-to-1 correspondence between our two classes of linear functions. Note that the Fourier basis functions \( \chi_S : \{0,1\}^n \to \{-1,1\} \) are linear functions in the sense immediately above.

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 this can be inefficient. We 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 \( f(x + y) \).
This algorithm was analyzed by [blr-03] as follows.

**Theorem 21.6.** Let $f : \{0,1\}^n \to \{-1,1\}$. Then
\[
\min_S \mathbb{P}_x[f(x) \neq \chi_S(x)] \leq \mathbb{P}_{x,y}[f(x)f(y) \neq f(x+y)],
\]
where $x, y \in \{0,1\}^n$ are distributed uniformly and independently over $\{0,1\}^n$.

**Proof.** For ease of notation, let $P = \mathbb{P}_{x,y}[f(x)f(y) \neq f(x+y)]$. Let $Z \in \{0,1\}$ be the indicator variable for the event that $f(x+y) \neq f(x)f(y)$. We have $P = \mathbb{E}[Z]$. We also have
\[
Z = 1 - \frac{1}{4}(f(x)f(y) - f(x+y))^2 \overset{(a)}{=} 1 - \frac{1}{4}(2 - 2f(x)f(y)f(x+y)) = \frac{1}{2}(1 + f(x)f(y)f(x+y)),
\]
where (a) observes that $f^2(x) = f^2(y) = f^2(x+y) = 1$. Thus
\[
P = \mathbb{E}[Z] = \frac{1}{2} + \mathbb{E}_x[f(x)\mathbb{E}_y[f(y)f(x+y)]] = \frac{1}{2} + \frac{1}{2}\langle f, h \rangle_b \overset{(b)}{=} \frac{1}{2} + \frac{1}{2}\langle \hat{f}, \hat{h} \rangle_b,
\]
where we define $h(x) = \mathbb{E}_y[f(y)f(x+y)]$. (b) applies the Fourier transform. We claim that $\hat{h}_S = \hat{f}_S^2$ for all $S$. Indeed, we have
\[
\mathbb{E}_x[h(x)\chi_S(x)] = \mathbb{E}_{x,y}[f(y)f(x+y)\chi_S(x)] \overset{(c)}{=} \mathbb{E}_{x,y}[f(y)f(x+y)\chi_S(y)\chi_S(x+y)] = \mathbb{E}_y[f(y)\chi_S(y)]\mathbb{E}_{x,y}[f(x+y)\chi_S(x+y)] = \langle f, \chi_S \rangle_b^2.
\]
(c) is by linearity of $\chi_S$. (d) observes that $y$ and $x+y$ are independently and uniformly distributed in $\{0,1\}^n$. Plugging back in, we now have
\[
P = \frac{1}{2} + \frac{1}{2}\sum_S \hat{f}_S^2 \leq \frac{1}{2} + \frac{1}{2}\max_S \hat{f}_S
\]
(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
\[
4\mathbb{P}_x[f(x) \neq \chi_S] = \|f - \chi_S\|_b^2 = \|f\|_b^2 + \|\chi_S\|_b^2 - 2\langle f, \chi_S \rangle = 2 - 2\hat{f}_S \leq 4P,
\]
as desired. \qed

Note that 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 $\chi_S$ that agrees with $f$ one hundred percent of the time. Thus we deduce the following.

**Corollary 21.7.** All linear functions $f : \{0,1\}^n \to \{-1,1\}$ are of the form $\chi_S$ for some set $S$. 

21.3.2 Locally correcting for linearity

**Theorem 21.8.** Let \( f : \{0, 1\} \rightarrow \{-1, 1\} \) be \( \epsilon \)-close to a basis function \( \chi_S : \{0, 1\} \rightarrow \{-1, 1\} \). 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

\[
P_y[f(y)f(x+y) = \chi_S(x)] \geq 1 - 2\epsilon.
\]

*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 \( \chi_S(x) = \chi_S(x+y)\chi_S(y) \).

21.3.3 A remark on convolutions

A key component of the proof of theorem 21.6 is the identity \( \hat{h}_S = \hat{f}_S^2 \) for the function \( h(x) = \mathbb{E}_y[f(y)f(x+y)] \). More generally, for two boolean formulas \( f, g : \{0, 1\} \rightarrow \mathbb{R} \), the convolution of \( f \) and \( g \), denoted \( f \ast g \), is the function defined by

\[
(f \ast g)(x) = \mathbb{E}_y[f(x)g(x+y)].
\]

The following identity is called *Plancheral’s identity* and generalizes the calculations used in theorem 21.6.

**Lemma 21.9.** Let \( f, g : \{0, 1\} \rightarrow \mathbb{R} \). Then \( \hat{(f \ast g)}_S = \hat{f}_S\hat{g}_S \) for all \( S \subseteq [n] \).

We leave the proof as an exercise.

21.4 Dictators

A function \( f : \{0, 1\}^n \rightarrow \{-1, 1\} \) is a *dictator* if it is one of the singleton basis functions\(^2\),

\[
f = \chi_i \text{ for some } i \in [n].
\]

The main goal in this section is to design a test for whether or not a function \( f : \{0, 1\}^n \rightarrow \{-1, 1\} \) is a dictator function. The test we design will be a composition of two tests. First, clearly, any dictator function is linear, which gives our first test.

---

\(^2\)For ease of notation, we write \( \chi_i \) instead of \( \chi_{\{i\}} \).
1. **Linearity test:** Sample \( x, y \in \{0, 1\}^n \) independently and reject if \( f(x + y) \neq f(x)f(y) \).

Our second test will be a new one. 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 \( \Omega^n \) to denote the triplets of vectors \( x, y, z \in \{0, 1\}^n \) such that for all \( i \), \((x_i, y_i, z_i) \in \Omega \). We note that to sample a uniformly random \((x, y, z) \in \Omega^n\), one can independently sample, for each \( i \in [n] \), three coordinates \((x_i, y_i, z_i) \in \Omega\) uniformly at random. We write \((x, y, z) \sim \Omega^n\) to denote \((x, y, z) \in \Omega^n\) sampled uniformly at random.

Observe that 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 gives 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 21.10.** 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 or error \( \leq \epsilon \) for \( \epsilon \leq .1 \). Then there exists a coordinate \( i \in [n] \) such that

\[
P_x[f(x) \neq \chi_i(x)] \leq \epsilon,
\]

where \( x \in \{0, 1\}^n \) is sampled uniformly at random.

To prove theorem 21.10, we first require the following lemma analyzing the not-all-equal test.

**Lemma 21.11.** Let \( f : \{0, 1\}^n \to \{-1, 1\} \). Let \((x, y, z) \sim \Omega^n \). Then

\[
P[(f(x), f(y), f(z)) \in \Omega] \leq \frac{7}{9} + \frac{2}{9} \sum_{i=1}^{n} \hat{f}_i^2.
\]

We will prove this lemma below in section 21.4.3. First, let us use it to prove theorem 21.10.

**Proof of theorem 21.10.** 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

\[
1 = \|\hat{f}\|^2 \geq 1 - 4.5\epsilon + (1 - 2\epsilon)^2 > 1,
\]

293
a contradiction. Thus \( \hat{f}_i \geq 1 - 2\epsilon \) for some \( i \). Then

\[
4 \mathbb{P}_x [f(x) \neq \chi_i(x)] \overset{\text{def}}{=} \|f - \chi_i\|_b^2 = \|\hat{f} - \hat{\chi}_i\|^2 \overset{\text{(a)}}{=} (\hat{f}_i - 1)^2 + 1 - \hat{f}_i^2 = 2 - 2\hat{f}_i \leq 4\epsilon,
\]

as desired. (a) is by lemma 21.4. (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 increasing the error rate with a simple trick, as follows.

**Theorem 21.12.** 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 \( \epsilon \)-far from any dictator function and \( \epsilon \leq 0.2 \), 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 \( p \leq 0.05 \), \( f \) is at most \( 4p \)-far from some dictator.

**21.4.1 Subclasses 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 21.10.
2. With probability 1/2, run the locally correcting protocol for linear functions for \( f \) with input string \( 1_S \), accepting \( f \) if this protocol returns 1.

This test has the following bounds.
Theorem 21.13. 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 \( \epsilon \)-far from any function in \( D_S \), then the test rejects \( f \) with probability \( \geq c\epsilon \) 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 \( \chi_i \) and in particular from a linear function, the correction protocol returns \( \chi_i(1_S) \) 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 \( \chi_i \) for some \( i \in \chi_i \).

21.4.2 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 \( T_p f : \{0, 1\}^n \to \mathbb{R} \) by

\[
(T_p f)(x) = \mathbb{E}_{y \sim \mathcal{N}_p(x)}[f(y)].
\]

Lemma 21.14. Let \( f : \{0, 1\} \to \mathbb{R} \) be a boolean function. For \( S \subseteq [n] \), \( (T_p f)_S = (1 - 2p)^{|S|} \hat{f}_S \).

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

\[
(T_p \chi_S)(x) = \mathbb{E}_{y \sim \mathcal{N}_p(x)}[\chi_S(y)] = \prod_{i \in S} \mathbb{E}_{y \sim \mathcal{N}_p(x)}[(-1)^{y_i}] = \prod_{i \in S}((1 - p)(-1)^{x_i} - p(-1)^{x_i})
\]

\[
= \prod_{i \in S}(1 - 2p)(-1)^{x_i} = (1 - 2p)^{|S|}\chi_S(x),
\]
as desired.  

295
21.4.3 Analysis of the not-all-equals test

Finally, let us analyze the not-all-equals test and prove lemma 21.11.

**Lemma 21.11.** Let \( f : \{0, 1\}^n \to \{-1, 1\} \). Let \((x, y, z) \sim \Omega^n\). Then

\[
P[(f(x), f(y), f(z)) \in \Omega] \leq \frac{7}{9} + \frac{2}{9} \sum_{i=1}^{n} \hat{f}_i^2.
\]

**Proof.** Let \( P = P[(f(x), f(y), f(z)) \in \Omega] \). Define a boolean function \( \text{NAE} : \{-1, 1\}^3 \to \{0, 1\} \) by setting

\[
\text{NAE}(a, b, c) =
\begin{cases}
0 & \text{if } a = b = c, \\
1 & \text{otherwise}.
\end{cases}
\]

We have

\[
\text{NAE}(a, b, c) = \frac{1}{8}((a - b)^2 + (a - c)^2 + (b - c)^2) = \frac{1}{8}(2a^2 + 2b^2 + 2c^2 - 2ab - 2ac - 2bc)
\]

\[
= \frac{3}{4} - \frac{1}{4}(ab + ac + bc).
\]

Thus

\[
P = \mathbb{E}[\text{NAE}(f(x), f(y), f(z))] = \frac{3}{4} - \frac{1}{4} \mathbb{E}[f(x)f(y) + f(y)f(z) + f(x)f(z)]
\]

\[
\overset{(a)}{=} \frac{3}{4} \mathbb{E}[f(x)f(y)]
\]

where (a) is by symmetry of \( \Omega \). Consider \( \mathbb{E}[f(x)f(y)] \). We have

\[
\mathbb{E}[f(x)f(y)] \overset{(b)}{=} \mathbb{E}_{x,y \sim N_{2/3}(x)}[f(x)f(y)] = \langle f, T_{2/3} f \rangle_b \overset{(c)}{=} \langle \hat{f}, (T_{2/3} \hat{f}) \rangle
\]

\[
= \sum_S (-1/3)^{|S|} \hat{f}_S^2 \geq -\frac{1}{3} \sum_i \hat{f}_i^2 - \frac{1}{27} \sum_{|S| \geq 3, \text{odd}} \hat{f}_S^2
\]

\[
\overset{(d)}{=} -\frac{1}{3} \sum_i \hat{f}_i^2 - \frac{1}{27} \left( 1 - \sum_i \hat{f}_i^2 \right)
\]

\[
= -\frac{1}{27} - \frac{8}{27} \sum_i \hat{f}_i^2.
\]

Here (b) observes that \( x \) is sampled uniformly from \( \{0, 1\}^n \), and conditional on \( x \), \( y \) is distributed as \( N_{2/3}(x) \). (c) applies the unitary Fourier transform. (d) is by lemma 21.14. (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. \( \square \)
21.5 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 21.1. We remind the reader that the following theorem is a critical component of the proof of the PCP theorem by Dinur, as discussed in section 21.1.

**Theorem 21.1.** Let \( L \subseteq \{0,1\}^n \). Let \( p = 2^{2n} \). Then there is a randomized algorithm \( A_L : \{0,1\}^n \times \{0,1\}^p \rightarrow \{0,1\} \) that, given oracle access to \( x \in \{0,1\}^n \) and \( y \in \{0,1\}^p \), has the following properties.

1. \( A_L \) makes 3 queries to bits of \( x \) or \( y \).
2. If \( x \in L \), then there exists \( y \in \{0,1\}^p \) such that \( A_L(x,y) = 1 \) always.
3. If \( x \notin L \) then for all \( y \in \{0,1\}^p \), we have
   \[
   P[A_L(x,y) = 0] \geq .001 \min_{y \in L} \frac{\|x - y\|_0}{n}.
   \]

**Proof.** We briefly described the algorithm in section 21.1 but let us describe it anew and more precisely. Let \( N = 2^n \). Identifying \( \{0,1\}^n \equiv [N] \), we identify \( L \) as a subset of \([N]\). Consider the subclass of dictator functions on \( N \) bits, \( D_L = \{\chi_w : \{0,1\}^N \rightarrow \{0,1\} \mid w \in L\} \).

Alternatively, given the subclass of dictators \( D_L \), we have a language \( L \subseteq \{0,1\}^n \) where \( x \in L \) iff \( \chi_x \in D_L \). The advantage of interpreting \( x \) as the index of a dictator \( \chi_x \), and interpreting \( L \) as the subclass of dictators \( D_L \), is that we have by now developed powerful tests for Boolean functions such as \( \chi_x \), and for families of dictators such as \( D_L \).

Given \( x \in \{0,1\}^n \), a proof for \( x \in L \) will be the (encoding of the) \( x \)th dictator function \( \chi_x : \{0,1\}^N \rightarrow \{-1,1\} \), as a length \( N \) bit string. Given input \( x \in \{0,1\}^n \) and candidate proof \( Y \in \{0,1\}^N \), we will test for two things.

1. We test that \( Y \in D_L \), using the test from theorem 21.13.
2. Given that \( Y = \chi_w \in D_L \) for some coordinate \( w \in L \), (somehow) test that \( w = x \).
We need to specify how to do the second step. Given that \( Y = \chi_w \) for some \( w \), we would like to check that \( w_j = x_j \) for a random coordinate \( j \in [n] \). We can query \( x_j \) but we cannot directly query \( w_j \). The dictator test in step 1 tells us that some \( w \) (probably) exists, but does not specify which \( w \).

Fix a coordinate \( j \). Recall the linear correction protocol from section 21.3.2. Insofar as \( Y \) is close to \( \chi_w \), we can probabilistically query \( \chi_w(Z) \) for our choice of input \( Z \in \{0,1\}^N \). To retrieve \( w \) without knowing \( w \), we need to define an input \( Z \in \{0,1\}^n \) such that \( \chi_w(Z) = w_j \) for all \( w \). To that end, we define a string \( Z_j \in \{0,1\}^N \) by

\[
Z_j(w) = w_j.
\]

For input \( Z \in \{0,1\}^n \), let \( H(Y,Z) = Y(Z + A)Y(A) \) (where \( A \sim \{0,1\}^N \)) denote the (random) output of running the local correction procedure on \( Y \) with input \( Z \). We reject \( Y \) unless \( x_j = H(Y,Z_j) \). If we inline the correction protocol of section 21.3.2, then step 2 can be written out explicitly as follows.

2. Sample \( j \in [n] \) uniformly at random and sample \( A \in \{0,1\}^N \) uniformly at random. Define \( Z_j \) by \( Z_j(w) = w_j \) for \( w \in \{0,1\}^n \). Reject \( x \) unless \( x_j = Y(Z_j + A)Y(A) \).

Having now established the testing algorithm in full, fix an input \( x \in \{0,1\}^n \) and \( Y \in \{0,1\}^N \), and suppose the tester accepts with probability of error \( \epsilon \) for \( \epsilon > 0 \). This means in particular that \( (x,Y) \) would pass either of the two tests alone with probability of error \( \leq 2\epsilon \). We claim that \( x \) is \( O(\epsilon) \)-close to some point in \( L \).

Because \( Y \) passes the first test with probability of error \( \leq 2\epsilon \), we have that \( Y \) is \( (C_1\epsilon) \)-close to some dictator function \( \chi_w \in D_L \), for some universal constant \( C_1 \). Since \( Y \) is \( (C_1\epsilon) \)-close to \( \chi_w \) for some \( w \), for any input \( Z \),

\[
P[H(Y,Z) \neq \chi_w(Z)] \leq C_2\epsilon \tag{21.2}
\]

for another universal constant \( C_2 > 0 \). Consider now the inputs \( Z_j \) that are constructed as a function of the randomly selected coordinate \( j \). We have

\[
P_{j}[x_j \neq w_j] \leq P_{j,H}[H(Y,Z_j) \neq w_j] + P_{j,H}[H(Y,Z_j) \neq x_j] \leq C_2\epsilon + 2\epsilon.
\]

(a) is by the union bound. The first term in (b) is by (21.2) and the second term is because \( (x,Y) \) passes the second test with probability of error \( \leq 2\epsilon \). Thus for a universal constant \( C_3 = C_2 + \epsilon \), \( x \) is \( (C_3\epsilon) \)-close to some \( w \in L \).
Appendix A

Homework assignments
Homework 0

- Due 11:59PM on Wednesday, August 31.
- Please be aware of appendix C.4 (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 1.1–1.10 as warmup exercises, especially if probability theory is new for you.

1. Exercise 1.11.
2. Exercise 1.12.
Homework 1

• Due 11:59PM on Friday, September 16.
• Please be aware of appendix C.4 (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 2.2
2. Exercise 2.3
3. Exercise 2.4 (This one is the trickiest, IMO.)
4. Exercise 3.2
5. Exercise 3.3
Homework 2

- Due 11:59PM on Wednesday, September 28.
- Please be aware of appendix C.4 (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 4.2
2. Exercise 5.1
3. Exercise 6.3
4. Exercise 6.4 (This one is the trickiest, IMO.)
5. Exercise 7.5
Appendix B

Selected solutions

Exercise 1.11. For $k \in \mathbb{N}$, 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? Prove your answer.

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 1.11.

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

   \[ E[X \mid \text{first coin is tails}] = 1 + E[X]. \]

   Therefore

   \[ E[X] = p + (1 - p)(1 + E[X]) = 1 + (1 - p)E[X]. \]

   This is solved by

   \[ E[X] = \frac{1}{p}. \]

2. (See part 2 below.)

---

1If 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 $p = 1/2$. 

303
3. Let $X_i$ be the number of flips between the $(i - 1)$th heads and the $i$th heads. Then $X_1 + \cdots + X_k$ is the total number of flips until $k$ heads. We have

$$E[X_1 + \cdots + X_k] = E[X_1] + \cdots + E[X_k] = \frac{k}{p}.$$ 

**Exercise 1.12.** Recall the quick-select algorithm introduced in section 1.1.3. The first goal of this exercise is to prove that quick-select takes $O(n)$ time in expectation. We ask you to prove this in two different ways which offer two different perspectives. Both analyses should use linearity of expectation and we ask you to point this out for both.

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 $O(n)$. Use this to obtain a $O(n)$ expected running time, overall.

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 $k$th 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 $O(1)$.\(^2\)

   (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 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 $O(n)$.

\(^2\)Hint: Exercise 1.11.
Solution to exercise 1.12.

Part 1. The running time, up to constants, is given by \( \sum_{i<j} X_{ij} \). The expected running time is given by \( E\left[\sum_{i<j} X_{ij}\right] \). By (a) linearity of expectation, we have

\[
E\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} E[X_{ij}]
\]

Each sum in the RHS above represents one of the four cases. Below, we show that each sum is at most \( O(n) \).

(a) \( i < j < k \): \( i \) and \( j \) are compared iff either is chosen as a pivot before any other element with rank between \( i \) and \( k \). That is,

\[
E[X_{ij}] = \frac{2}{k - i + 1}.
\]

We have

\[
\sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} E[X_{ij}] = \sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} \frac{2}{k - i + 1} = \sum_{i=1}^{k-2} 2 \leq O(k) \leq O(n).
\]

(b) \( i < k < j \): \( i \) and \( j \) are compared iff either is selected as a pivot before any element between \( i \) and \( j \). That is,

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

Consider the sum

\[
\sum_{i=1}^{k-1} \sum_{j=k+1}^{n} E[X_{ij}] = \sum_{i=1}^{k-1} \sum_{j=k+1}^{n} \frac{2}{j - i + 1}
\]

For \( h \in [n] \), the term \( 2/(h + 1) \) appears at most \( \min\{k, h\} \) times (for \( i \) between \( \max\{k - h + 1, 1\} \) and \( k \)). Consequently

\[
\sum_{i=1}^{k-1} \sum_{j=k+1}^{n} \frac{2}{j - i + 1} \leq \sum_{h=1}^{n} \min\{k, h\} \cdot \frac{2}{h + 1} \leq O(n).
\]
(c) \( k < i < j \): \( i \) and \( j \) are compared iff either are selected as a pivot before any element between \( k \) and \( j \). That is,
\[
E[X_{ij}] = \frac{2}{j - k + 1}.
\]

Similar to the first case, we have
\[
\sum_{j=k+2}^{n} \sum_{i=k+1}^{j-1} E[X_{ij}] = \sum_{j=k+2}^{n} \sum_{i=k+1}^{j-1} \frac{2}{j - k - 1} = \sum_{j=k+2}^{n} O(1) \leq O(n),
\]
as desired.

(d) Either \( i = k \) or \( j = k \): \( i \) and \( j \) are compared iff either are selected as a pivot before any element between them. We have
\[
E[X_{ij}] = \frac{2}{j - i + 1}.
\]

We have
\[
\sum_{i=1}^{k-1} \frac{2}{k - i} + \sum_{j=k+1}^{n} \frac{2}{j - i} = O(\log n) \leq O(n),
\]
as desired.

**Approach 2. (Sketch)**

(a) Suppose there are \( n \) elements remaining and we want to seek the \( k \)th element. With probability \( 1/2 \), we pick from one of the middle \( \lceil n/2 \rceil \) elements, in which case we either identify the \( k \)th element, or throw out at least \( n/4 \) elements.

(b) Suppose the number of elements remaining is in the range \([4/3)^i, (4/3)^{i+1}\]). By exercise 1.11, it takes \( O(1) \) iterations in expectation to pick from one of the middle \( \lceil n/2 \rceil \) elements, in which case we throw out at least \( 1/4 \)th of the elements, and the total number of elements drops out of the range.

(c) Let \( X_i \) be the expected number of iterations where the number of elements is in the range \([4/3)^i, (4/3)^{i+1}\]). By part (b), \( E[X_i] = O(1) \). Each such iteration takes \( O((4/3)^i) \) time. The expected time over all such iterations (up to constants) is given by
\[
E[(4/3)^i X_i] = (4/3)^i E[X_i] = O((4/3)^i),
\]
as desired.
(d) For \( i \in \mathbb{Z}_{\geq 0} \), let \( T_i \) be the total time spent where the total number of elements lies in the range \([(4/3)^i, (4/3)^{i+1}]\). By part (c), we have
\[
E[T_i] = O\left((4/3)^i\right)
\]
for \( i \leq \log_{4/3}(n) \). In total the expected running time is given by
\[
E\left[\sum_{i=1}^{\left\lfloor \log_{4/3}(n) \right\rfloor} (4/3)^i\right] = O\left((4/3)^{\log_{4/3}(n)}\right) = O(n),
\]
as desired.

Exercise 1.13. 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 \( O(n^{2.37...}) \). 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 \( A(Bx) \) then \( AB \).) If they are unequal, then certainly \( AB \neq C \) and we output \text{false}. Otherwise we output \text{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 least \( 1/2 \).

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 \( \langle x, y \rangle \overset{\text{def}}{=} \sum_{i=1}^{n-1} x_i y_i \neq 0 \) with probability at least \( 1/2 \).  

2. Use the preceding result to show that if \( AB \neq C \), then with probability at least \( 1/2 \), \( ABx \neq Cx \).

3. Suppose we want to decrease our probability of error to (say) \( 2^{-n} \). Based on the algorithm above, design and analyze fast randomized algorithm with the following guarantees.

   • If \( AB = C \), then it always reports that \( AB = C \).
   
   • If \( AB \neq C \), then with probability at least \( 1 - 2^{-n} \), it reports that \( AB \neq C \).

   \(^3\text{Hint: Suppose for simplicity that the last coordinate of } y \text{ is nonzero. It might help to imagine sampling the first } n - 1 \text{ bits and computing the partial sum } S_{n-1} = \sum_{i=1}^{n-1} x_i y_i \text{ first, before sampling } x_n \text{ and adding } x_n y_n. \text{ Formally your analysis may involve some conditional probabilities. (And what about the case where } y_n = 0? \text{)}\)

   \(^4\text{Even if you haven’t solved part 1 you may assume it to be true.}\)
(Your analysis should include the running time as well.)\(^5\)

**Solution to exercise 1.13.**

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 \( \alpha \). 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
\[
\Pr[\langle x, y \rangle = 0] = \Pr[\langle x, y \rangle = 0 | \alpha = 0] \Pr[\alpha = 0] + \Pr[\langle x, y \rangle = 0 | \alpha = 1] \Pr[\alpha = 1] \\
\geq \Pr[x_k = 1] \Pr[\alpha = 0] + \Pr[x_k = 0] \Pr[\alpha = 1] \frac{1}{2} \Pr[\alpha = 0] + \frac{1}{2} \Pr[\alpha = 1] = \frac{1}{2},
\]
as desired.

2. If \( AB \neq C \), then there is some row \( i \) such that the \( i \)th row of \( AB \) does not equal the \( i \)th row of \( C \). Recall that \( (ABx)_i \) and \( (Cx)_i \) are the inner product of \( x \) with the \( i \)th 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. We repeat the algorithm from part 2 \( n \) 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 \( x \) trials is therefore \( 1/2^n \). The total running time is \( O(n^3) \), since it takes \( O(n^2) \) time to compute \( ABx \) and \( Cx \), and we rerun the algorithm \( n \) times.

**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 \( h : [n] \rightarrow [k] \) as follows. First, let \( p \) be any prime number \( > n \). Draw \( a \in \{1, \ldots, p-1\} \) uniformly at random, and draw \( b \in \{0, \ldots, p-1\} \) 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 \( c_1, c_2 \in \{0, \ldots, p-1\} \) with \( c_1 \neq c_2 \). Show that the system of equations
\[
ax_1 + b = c_1 \mod p \\
ax_2 + b = c_2 \mod p
\]
uniquely determines \( a \in \{1, \ldots, p-1\} \) and \( b \in \{0, \ldots, p-1\} \).\(^6\)

---

\(^5\)Note: I should have written “probability of error to (say) 1/n^2”, instead of 2^{-n}, because the running time for 1/n^2 error probability is more interesting. But I will leave it as it is since we are close to the deadline.

\(^6\)Here it is helpful to know that division is well-defined on the set of integers modulo \( p \) when \( p \) is prime. More precisely, “\( a/b \)” is defined as the unique integer \( c \) such that \( bc = a \).
B. Selected solutions

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

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

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

(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 \(c_1 \in \{0, \ldots, p-1\}\). Show that

\[
\sum_{\substack{c_2 \in \{1, \ldots, p\} \\text{such that}\ \ 0 \neq c_1 - c_2 \mod k}} P[ax_1 + b = c_1, ax_2 + b = c_2] \leq \frac{1}{pk}.
\]

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

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

Solution to exercise 2.2.

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 \(ax_1 + b = c_1\) and \(ax_2 + b = c_2\). The probability of sampling these particular values of \(a\) and \(b\) is \(1/p(p-1)\).

---

7 Here we note that for \(x_1 \neq x_2\), \(ax_1 + b = ax_2 + b_2 \mod p\) iff \(a = 0\).
8 There was a typo were previously the RHS said “1/n” instead of “1/pk”.
9 Hint: You may want to show that the number of values \(c_2 \in [p]\) such that \(c_1 = c_2 \mod k\) is \(\leq \frac{p-1}{n}\).
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 $[p/k]$ 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
\[
\sum_{\substack{c_2 \in \{1, \ldots, p\} \\
\, c_1 = c_2 \mod k \land c_2 \neq c_1}} \mathbb{P}[ax_1 + b = c_1, ax_2 + b = c_2] \leq \frac{p - 1}{k} \cdot \frac{1}{p(p - 1)} = \frac{1}{pk}.
\]

4. We have
\[
\mathbb{P}[h(x_1) = h(x_2)] = \sum_{c_1 \in [p]} \mathbb{P}[ax_1 + b = c_1 \text{ and } h(x_1) = h(x_2)] \leq \sum_{c_1 \in [p]} \frac{1}{pk} = \frac{1}{k}.
\]

Here (b) is by part 3.

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

1. Contains all of the $\epsilon$-heavy hitters.
2. Only includes $(\epsilon/2)$-heavy hitters.

Your space usage should be comparable to the space used by the count-min-sketch data structure.

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 $\epsilon$-heavy hitters, and (b) only includes $(\epsilon/2)$-heavy hitters. The tricky part is that count-min-sketch only approximates the frequencies. You may want to account for the fact that an instance of count-min-sketch($\epsilon, \delta$) may overestimate the relative frequency of an element by as much as $\epsilon$, which can make a very infrequent element look like an $\epsilon$-heavy hitter.

---

\[10\] Here the elements are integers from $[n] = \{1, \ldots, n\}$, where $n$ is known, and $\epsilon \in (0, 1)$ is an input parameter.

\[11\] 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.

\[12\] You may want to use the count-min($\epsilon, \delta$) data structure as a black box, but you should be clear about your choice of parameters $\epsilon$ and $\delta$. 

310
Solution to exercise 2.3. (Sketch) We maintain

1. An instance of the count-min-sketch with additive error $\epsilon/2$ and error parameter $\delta = 1/n^3$. (Larger polynomials would be fine.)

2. A counter recording the total length $m$ of the data structure.

3. The set of elements in a min-heap $S$.

4. The frequencies of all the elements.

Whenever an element $i$ is presented by the stream, we first increase the estimated frequency the count-min data structure. We then query the data structure and receive an estimate $\tilde{f}_i$ of the frequency of $i$.

If the frequency estimate is at least $\epsilon m$, then we either insert the element into the heap, or update its value in the heap.

We also remove form the heap any element whose estimated frequency falls below $\epsilon m$.

To summarize, every element in $S$ has estimated sequence less than $\epsilon m$.

We briefly justify the correctness. Fix a point in my the stream. For each element $i \in [n]$, the probability that we overestimate the frequency by more than $\epsilon m/2$ is at most $1/n^3$. Taking the union bound over all $[n]$, the probability we overestimate any of the frequencies by more than $\epsilon m/2$ is at most $1/n^2$.

Suppose such is the case. If an element $e$ has relative frequency at least $\epsilon$ (now), then it had relative frequency at least $\epsilon$ when it last appeared in the stream (when $m$ was no bigger). At that point $e$ must be inserted into $S$. Since $e$ also has relative frequency at least $\epsilon$ between now and then, it is never removed from $S$. Thus $S$ contains all the $\epsilon$-heavy hitters.

Now let $e \in S$. Then $e$ has estimated relative frequency at least $\epsilon$. Since this overestimates the true relative frequency is by at most $\epsilon/2$, $e$ is an $(\epsilon/2)$-heavy hitter. That is, $S$ only contains $(\epsilon/2)$-heavy hitters.

count-min spends $O(\log n)$ time per element to update $O(\log n)$ counters, and the heap takes $O(\log n)$ time per insertion and deletion. We can amortized deleting an element against inserting the element earlier. So the data structure takes $O(\log n)$ amortized time per element.

Exercise 2.4. In this exercise, we develop a refined analysis that can reduce the additive error substantially in many real settings.

Let $S$ denote the sum of frequency counts of all elements that are not $\epsilon$-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 \( w \) by a constant factor, and using \( k \)-wise independent hash functions for \( k = O(1/\epsilon) \) (see page 55 for the definition of \( k \)-wise independent hash functions), one can estimate the frequency of every element with additive error at most \( \epsilon S \) with high probability in \( O(\log(n)/\epsilon) \) space.\(^{13}\)\(^{14}\)\(^{15}\)

**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

\[
P[\text{at least } \ell \text{ items being hashed to } A[i]] \leq \frac{1}{\ell!}.
\]

2. Show that, with probability of error \( \leq 1/n^2 \), the maximum length is at most \( O(\log(n)/\log \log n) \).\(^{16}\)

**Solution to exercise 3.2.**

**Part 1.** By taking the union bound over all sets of \( \ell \) items landing in that bin, we have

\[
\binom{n}{\ell} \frac{1}{n^\ell} = \frac{n!}{\ell!(n-\ell)!n^\ell} \leq \frac{1}{\ell!}.
\]

**Part 2.** We have

\[
\frac{1}{\ell!} \leq \left( \frac{1}{(\ell/2)} \right)^{\ell/2} \leq 2^{-(\ell/2) \log(\ell/2)}.
\]

Now, for \( \ell = 16 \log(n)/\log \log n \)

\[
\frac{\ell}{2} \log(\ell/2) = \frac{8 \log(n) \log(8 \log n/\log n)}{\log \log n} = \frac{8 \log n (O(1) + \log \log n - \log \log \log(n))}{\log \log n} \geq (8 - o(1)) \log n
\]

\(^{13}\)Hint: It might be helpful to think about the special case of \( S = 0 \).

\(^{14}\)Hint: \( k \) was chosen to be greater than the maximum number of heavy hitters. Why?

\(^{15}\)Actually, there’s a way to do this with only universal hash functions. But you’re welcome to use \( k \)-wise independence if you find it convenient. You may first want to solve the problem assuming an ideal hash function, and figure out how to incorporate limited independence afterwards.

\(^{16}\)The simple lower bound of \( \ell! \geq (\ell/2)^{\ell/2} \) may be helpful. It is implicit in the \( O(\ldots) \) notation that your bound need only hold for \( n \) sufficiently large.
B. Selected solutions

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 \( h: [n] \rightarrow [k] \) be a universal hash function, with \( k \geq n \). 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 array 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 \( i \)th cell. We have

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

Show that\(^{17}\)

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

3. Show that

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

4. Show that

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

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.

\(^{17}\)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.
Solution to exercise 3.3.

**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

$$\left( \frac{n}{2} \right) \frac{1}{n^2} = \frac{n(n-1)}{2n^2} \leq \frac{1}{2}.$$

Thus the probability of having no collisions is at most $1/2$.

**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

$$\sum_i k_i^2 \leq n + \sum_{i:k_i>1} (k_i^2 - 1) \leq n + \sum_{i:k_i>1} 2 \left( \frac{k_i}{2} \right) = n + 2 \text{(total # of collisions)}.$$

**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

$$\left( \frac{n}{2} \right) \frac{1}{n} = \frac{n(n-1)}{2n} \leq \frac{n}{2}.$$

**Part 4.** We have

$$E[\text{total size}] \leq n + \sum_i E\left[ k_i^2 \right] \leq 2n + 2 E[\text{total # of collisions with respect to } h_0] \leq 3n.$$

By Markov’s inequality,

$$P[\text{total size } > 6n] \leq 1/2,$$

as desired.
Appendix C

CS381: 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 also be similar to the Fall 2020 course (cf. https://fundamentalalgorithms.com/randomized/f20).¹

Lectures are delivered in-person on

Tuesdays and Thursdays, from 4:30 to 5:45 PM, in Lawson Room 1106.

I will have office hours after class on both Tuesday and Thursday in my office, Lawson 1211. 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. I’m certain that slots will open up. You can add yourself to gradescope and submit homework in the meantime.

C.1 Textbooks

No textbook is strictly required as lecture notes² are provided. That said, the most closely aligned textbook with our course is:

- MotwaniRaghavan

Other books and monographs overlapping with the course include:

- AlonSpencer

¹Compared to Fall 2020, I may replace some topics that were covered in my Fall 2021 advanced graph algorithms course with new ones.
²The notes were first written in Fall 2020 to compensate for remote learning.
C. Syllabus, policies, and procedures

C.2. Correspondence

The following courses from other institutions also have lecture notes that you may find helpful.

- *Randomized algorithms*, taught by Sariel Har-Peled at UIUC. [https://sarielhp.org/teach/17/b/](https://sarielhp.org/teach/17/b/)
- *Algorithms for Big Data*, taught by Chandra Chekuri at UIUC. [https://courses.engr.illinois.edu/cs498abd/fa2020/](https://courses.engr.illinois.edu/cs498abd/fa2020/).

C.2 Correspondence

The course website is

[www.fundamentalalgorithms.com/randomized](http://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.

[piazza.com/purdue/fall2022/cs588/home](http://piazza.com/purdue/fall2022/cs588/home)

The point of Piazza is to foster discussion among the students. The TA will monitor Piazza on weekdays.

C.3 Grading

- 30% Homework
- 30% Midterm
We compute numerical scores based on the weighting above (as a fractional value between 0 and 1), and then we curve the grades.

C.4 Homework

This course has regular homework, generally due every two weeks (with the exception of homework 0.) Homeworks will be due at 11:59PM on Wednesday 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).

Gradescope. 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 V5BRK2.

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-third 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. For word problems, we have a simple late policy where you can submit up to a week late, at a cost of 35% 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.

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.

We plan to put up the solutions quickly after the homework is collected.

**Resubmitting homework.** You might have noticed that there is both a late policy and a plan to post solutions very soon after the submission deadline. You can take advantage of this by comparing the answer key to your own submission, and possibly resubmit your homework late even with the benefit of the answer key. If you do use the posted answer key in a resubmission, we expect you to cite it accordingly, and still express the solution in your own words.

**IDK.** One may simply write “I don’t know” or “IDK” and automatically get 25% of the possible points (for any problem or subproblem).

**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](http://overleaf.com) for typesetting LaTeX.

- Alternatively, the software **typora** 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**.

**C.5 On the COVID-19 pandemic**

This course may be impacted by the ongoing COVID-19 pandemic. Purdue has been very active about making the campus safe and more information can be found at the following url.

[https://protect.purdue.edu](https://protect.purdue.edu)

In particular, we are all expected to uphold the *Protect Purdue Pledge*.  

318
C.5.1 Quarantining

If you must quarantine or isolate at any point in time during the semester, please reach out to any of the staff (preferrable via a private Piazza note) so that we can communicate about how you can continue to learn remotely. Work with the Protect Purdue Health Center (PPHC) to get documentation and support, including access to an Academic Case Manager who can provide you with general guidelines/resources around communicating with your instructors, be available for academic support, and offer suggestions for how to be successful when learning remotely. Your Academic Case Manager can be reached at acmq@purdue.edu. Importantly, if you find yourself too sick to progress in the course, notify your academic case manager and notify a staff member. We will make arrangements based on your particular situation.

C.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.

C.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.
C.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](https://www.purdue.edu/provost/teachinglearning/honor-pledge.html).

C.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.

C.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).

C.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](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.

C.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](http://www.purdue.edu/ehps/emergencypreparedness/warning-system.html). Please review the Emergency Response Procedures at [https://www.purdue.edu/emergencypreparedness/flipchart/index.html](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/](http://www.purdue.edu/securepurdue/police/video/)

C.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.

C.14 Mental Health and Wellness

If you find yourself beginning to feel some stress, anxiety and/or feeling slightly overwhelmed, try WellTrack ([https://purdue.welltrack.com](https://purdue.welltrack.com)). Sign in and find information and tools at your fingertips, available to you 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](http://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.

C.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.
C.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).

C.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.

C.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.

C.19 Changes to the syllabus

This syllabus is subject to change and changes will be announced appropriately.