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 384) for more information.

Homework

- Homework 0 due August 31.
- Homework 1 due September 16.
- Homework 2 due September 28.
- Homework 3 due October 21.
- Homework 4 due November 4.
- Homework 5 due November 18.
- Homework 6 due December 9.

Class schedule¹

1. **August 23.** Randomized SAT and quicksort (chapter 1).
2. **August 25.** Hashing and heavy hitters (chapter 2).
3. **August 30.** Hash tables and linear probing (chapter 3).
4. **September 1.** Randomized minimum cut (sections 4.1–4.3).
5. **September 6.** Random graphs (chapter 5).
6. **September 8.** Randomized rounding (chapter 6).

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

8. September 15. Dimensionality reduction (chapter 8).


10. September 22. $L_1$-metric embeddings and sparsest cut (sections 10.1–10.3).

11. September 27. Randomized tree metrics (chapter 11).


(*) Midterm on Thursday, October 13, at 8PM in Hampton Hall, room 1252. (See appendix C.4.1.)

15. October 18. Lovász local lemma (chapter 15).


17. October 25. Discrepancy via Gaussian random walks (chapter 17).


(*) Thanksgiving break.


(*) Final on Thursday, December 15 at 8AM in Lawson, B151.
## Contents

**Basic information** 1

**Class schedule** 1

1 **Randomized searching and sorting** 11
   1.1 First randomized algorithms 11
   1.2 Basic probability 14
   1.3 Randomized approximations for SAT 21
   1.4 Randomized sorting 25
   1.5 Additional notes and materials 28
   1.6 Exercises 29

2 **Hashing and heavy hitters** 34
   2.1 Introduction 34
   2.2 Hashing 37
   2.3 Using hashing to approximate frequencies 38
   2.4 Amplification 40
   2.5 Extensions 41
   2.6 Takeaways 42
   2.7 Additional notes and references 43
   2.8 Exercises 44

3 **Hash tables and linear probing** 48
   3.1 Dictionaries 48
   3.2 Hash tables with chaining 52
   3.3 Linear probing 53
   3.4 4-wise independence 58
   3.5 Takeaways 61
   3.6 Additional notes and materials 61
   3.7 Exercises 61
## Contents

4 Sampling edges
- 4.1 Minimum cut ............................................. 64
- 4.2 Amplification by branching ............................... 69
- 4.3 Randomized branching .................................. 72
- 4.4 Sparsification .............................................. 73
- 4.5 Randomized Ford-Fulkerson .............................. 82
- 4.6 Additional notes and materials .......................... 87
- 4.7 Exercises .................................................. 87

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

6 Randomized Rounding
- 6.1 SAT ..................................................... 104
- 6.2 Set cover .................................................. 108
- 6.3 Covering integer programs ............................... 110
- 6.4 Additional notes and materials .......................... 113
- 6.5 Exercises .................................................. 113
- 6.6 Proof of the multiplicative Chernoff bound .......... 115

7 Distinct elements
- 7.1 Unique visitors ........................................... 118
- 7.2 Preliminaries on continuous random variables ........ 120
- 7.3 Distinct elements, continued ............................ 121
- 7.4 Variance and Chebyshev’s inequality .................. 124
- 7.5 Amplification: variance reduction and the median trick 124
- 7.6 Distinct elements with pairwise independent hash functions 128
- 7.7 Takeaways ................................................ 128
- 7.8 Additional notes and materials .......................... 130
- 7.9 Exercises .................................................. 130

8 Dimensionality Reduction
- 8.1 Large data sets and long vectors ....................... 133
- 8.2 Gaussian random variables: an interface .......... 135
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3</td>
<td>Random Projections</td>
<td>138</td>
</tr>
<tr>
<td>8.4</td>
<td>Gaussians</td>
<td>139</td>
</tr>
<tr>
<td>8.5</td>
<td>Additional notes and materials</td>
<td>142</td>
</tr>
<tr>
<td>8.6</td>
<td>Exercises</td>
<td>142</td>
</tr>
<tr>
<td>9</td>
<td>Locality sensitive hashing and approximate nearest neighbors</td>
<td>145</td>
</tr>
<tr>
<td>9.1</td>
<td>Nearest neighbor search</td>
<td>145</td>
</tr>
<tr>
<td>9.2</td>
<td>Locality Sensitive Hashing for Angles</td>
<td>146</td>
</tr>
<tr>
<td>9.3</td>
<td>Locality Sensitive Hashing for Euclidean Distance</td>
<td>148</td>
</tr>
<tr>
<td>9.4</td>
<td>Additional notes and materials</td>
<td>153</td>
</tr>
<tr>
<td>9.5</td>
<td>Exercises</td>
<td>153</td>
</tr>
<tr>
<td>10</td>
<td>$L_1$-metric embeddings and sparsest cut</td>
<td>155</td>
</tr>
<tr>
<td>10.1</td>
<td>LP duality, line embeddings, and max-flow min-cut</td>
<td>155</td>
</tr>
<tr>
<td>10.2</td>
<td>Sparsest cut</td>
<td>164</td>
</tr>
<tr>
<td>10.3</td>
<td>Rounding via $L_1$-metric embeddings</td>
<td>167</td>
</tr>
<tr>
<td>10.4</td>
<td>Rounding via region growing (for uniform demands)</td>
<td>175</td>
</tr>
<tr>
<td>10.5</td>
<td>Application: Minimum bisection</td>
<td>178</td>
</tr>
<tr>
<td>10.6</td>
<td>Additional notes and materials</td>
<td>179</td>
</tr>
<tr>
<td>10.7</td>
<td>Exercises</td>
<td>180</td>
</tr>
<tr>
<td>11</td>
<td>Tree Metrics</td>
<td>183</td>
</tr>
<tr>
<td>11.1</td>
<td>Introduction</td>
<td>183</td>
</tr>
<tr>
<td>11.2</td>
<td>Low-Stretch Spanning Trees</td>
<td>184</td>
</tr>
<tr>
<td>11.3</td>
<td>Hierarchical Tree Metrics</td>
<td>187</td>
</tr>
<tr>
<td>11.4</td>
<td>Additional notes and materials</td>
<td>190</td>
</tr>
<tr>
<td>11.5</td>
<td>Exercises</td>
<td>190</td>
</tr>
<tr>
<td>12</td>
<td>Sampling geometric range spaces</td>
<td>192</td>
</tr>
<tr>
<td>12.1</td>
<td>Introduction</td>
<td>192</td>
</tr>
<tr>
<td>12.2</td>
<td>Proof of the $\epsilon$-sample theorem</td>
<td>195</td>
</tr>
<tr>
<td>12.3</td>
<td>VC dimension</td>
<td>197</td>
</tr>
<tr>
<td>12.4</td>
<td>Additional notes and materials</td>
<td>199</td>
</tr>
<tr>
<td>12.5</td>
<td>Exercises</td>
<td>199</td>
</tr>
<tr>
<td>12.A</td>
<td>Proof of the additive Chernoff bound</td>
<td>200</td>
</tr>
<tr>
<td>13</td>
<td>PAC learning</td>
<td>203</td>
</tr>
<tr>
<td>13.1</td>
<td>The consistency model</td>
<td>203</td>
</tr>
<tr>
<td>13.2</td>
<td>The PAC learning model</td>
<td>205</td>
</tr>
</tbody>
</table>
Contents

13.3 Generalization for finite hypothesis classes ................. 206
13.4 Occam’s razor ........................................... 207
13.5 Stronger generalization bounds via growth rate .............. 209
13.6 Additional notes and materials .................................. 211
13.7 Exercises ..................................................... 212

14 Entropy and error-correcting codes ....................... 213
14.1 Coding theory ............................................... 213
14.2 Entropy ......................................................... 218
14.3 Principle of Independence ...................................... 221
14.4 Additional notes and materials .................................. 222
14.5 Exercises ..................................................... 223

15 Lovász local lemma and resampling .................... 226
15.1 Resampling k-SAT ........................................... 226
15.2 The Lovász Local Lemma ..................................... 227
15.3 Analysis of the resampling algorithm ...................... 229
15.4 Additional notes and materials .................................. 233
15.5 Exercises ..................................................... 233

16 Random walks ................................................. 235
16.1 Ranking the web .............................................. 236
16.2 A linear map of the web ...................................... 239
16.3 Eigenvectors .................................................... 241
16.4 The Perron-Frobenius theorem .................................. 243
16.5 Perron-Frobenius for strongly connected random walks .... 246
16.6 Computing PageRank ......................................... 247
16.7 Additional notes and materials .................................. 248
16.8 Exercises ..................................................... 248

17 Discrepancy via Gaussian random walks ............. 249
17.1 Balanced coloring ............................................. 249
17.2 Reduction to partial coloring ................................ 250
17.3 Partial coloring by random walks: proof of lemma 17.2 .... 253
17.4 Additional notes and materials .................................. 258
17.5 Exercises ..................................................... 258
17.A Analyzing Gaussian martingales .......................... 259
18 Connectivity and Electricity  260
  18.1 Introduction .......................... 260
  18.2 Electrical networks ........................ 261
  18.3 Structure of electrical flows ........................ 263
  18.4 Effective resistance and the Laplacian ........................ 265
  18.5 Effective conductance ........................ 266
  18.6 Hitting times and cover times ........................ 267
  18.7 Additional notes and materials ........................ 270
  18.8 Exercises .......................... 270

19 Spectral analysis of undirected random walks  271
  19.1 The Laplacian of a graph ........................ 271
  19.2 The Spectral Theorem for Symmetric Maps ........................ 273
  19.3 Random walks in undirected graphs ........................ 275
  19.4 Additional notes and materials ........................ 278
  19.5 Exercises .......................... 279

20 Conductance  280
  20.1 Sparse cuts .......................... 280
  20.2 Conductance .......................... 281
  20.3 Proving $\Psi(G) \leq \sqrt{2\lambda_2}$ ........................ 284
  20.4 Additional notes and materials ........................ 287
  20.5 Exercises .......................... 288

21 Deterministic log-space connectivity  289
  21.1 Introduction .......................... 289
  21.2 An overview of the deterministic connectivity algorithm ........................ 290
  21.3 Preliminaries .......................... 294
  21.4 Analysis of the zig-zag product ........................ 296
  21.5 Additional notes and materials ........................ 299
  21.6 Exercises .......................... 299

22 Reducing randomness with random walks  301
  22.1 Introduction .......................... 301
  22.2 High level overview: amplification by random walks ........................ 302
  22.3 Amplifying $\text{RP}$: proof of lemma 22.4 ........................ 304
  22.4 Efficiently amplifying $\text{BPP}$ ........................ 306
  22.5 Efficiently making large expanders ........................ 308
  22.6 Additional notes and materials ........................ 310
22.7 Exercises ................................................. 310

23 Randomized Proofs and Verification by Random Walks 311
  23.1 Randomized Proofs ..................................... 311
  23.2 Constraint satisfaction problems ..................... 313
  23.3 Graph CSP, and amplification ......................... 314
  23.4 Expander-ification .................................... 318
  23.5 Error amplification ................................... 319
  23.6 Alphabet reduction .................................... 325
  23.7 Additional notes and materials ....................... 328
  23.8 Exercises .............................................. 328

24 Randomly Testing Boolean Functions 331
  24.1 Testing boolean formulae with 3 queries, and 8 letter graph CSP’s 331
  24.2 Fourier analysis of boolean functions ................ 334
  24.3 Linearity .............................................. 336
  24.4 Dictators ............................................. 339
  24.5 Universal Tester ...................................... 343
  24.6 Additional notes and materials ....................... 345
  24.7 Exercises .............................................. 345

25 Online algorithms 346
  25.1 Caching .............................................. 346
  25.2 Buy-or-rent .......................................... 350
  25.3 Additional notes and materials ....................... 354
  25.4 Exercises .............................................. 354

26 Sparsification 356
  26.1 Graph sparsification ................................ 356
  26.2 Spectral sparsification ................................ 358
  26.3 Exercises .............................................. 359

A Homework assignments 373
  Homework 0 .............................................. 374
  Homework 1 .............................................. 375
  Homework 2 .............................................. 376
  Homework 3 .............................................. 377
  Homework 4 .............................................. 378
  Homework 5 .............................................. 379
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 \((x_1, x_2, x_3) = (t, f, t)\).

In the 3-SAT problem, we are given a 3-SAT formula \(f(x_1, \ldots, x_n)\) with \(m\) clauses and \(n\) variables and want to find a satisfying assignment.

3-SAT is the quintessential NP-Complete search problem, and there is no polynomial time algorithm to solve it unless \(P = NP\) [Coo71; Lev73]. 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

\[ ^1 \text{Here } t \text{ denotes “true” and } f \text{ denotes “false”} \]
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 \( \left( \frac{7}{8} \right) m \) 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 \( \left( \frac{7}{8} \right) m \) 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 \( \left( \frac{7}{8} + \epsilon \right) \)-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 \( \left( \frac{7}{8} \right) \) th of the clauses. Moreover, for all \( \epsilon > 0 \), a polynomial time approximation algorithm with approximation ratio \( \left( \frac{7}{8} + \epsilon \right) \) implies that \( P = NP \).
1. Randomized searching and sorting

1.1. First randomized algorithms

Kent Quanrud  
Fall 2022

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. 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. Randomized searching and sorting

1.2. Basic probability

```plaintext
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 ∈ [n] uniformly at random.
2. Compute the rank ℓ of A[i]. // O(n)
3. If ℓ = k, then return A[i].
4. If ℓ > k, then recursively search for the rank k element among the set of ℓ − 1 elements less than A[ℓ], and return it.
5. If ℓ < k, then recursively search for the rank k − ℓ element among the set of n − ℓ elements greater than A[ℓ], and return it.
```

Figure 1.3: A randomized algorithm for selection.

1.1.3 Selection.

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

The obvious solution is to sort the list, which takes $O(n \log n)$ time. But in fact one can do better: the “median-of-medians” divide-and-conquer algorithm of Blum, Floyd, Pratt, Rivest, and Tarjan [BFPRT72] runs in $O(n)$ time. This algorithm is a bit tricky, both to describe and to analyze. Here then is a simpler alternative, which is similar to quick-sort: pick a pivot uniformly at random, and compute its rank ℓ. Depending on whether $k = ℓ$, $k < ℓ$, or $k > ℓ$, 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.** quick-select(A[1..n], k) returns the rank 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, we don’t know yet. Yet we can state without ambiguity that half the time it will land heads, and half the time it will land tails. What do we mean when we say that half the time it will land heads? There is of course only one coin, and we can’t split the coin in half. 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: 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 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 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 $A$ not occurring. We always have

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

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

“$A \wedge B$” or “$A \cap B$” or “$A$ and $B$”.

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

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

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

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

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

\[ P[A \land B] \leq \min\{P[A], P[B]\} \]

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

\[ P[A \mid B] = P[A] \quad (1.2) \]

for two events \( A \) and \( B \). The equation above is equivalent to the one with \( A \) and \( B \) flipped, as well as

\[ P[A \land B] = P[A] P[B] \quad (1.3) \]

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

$$\text{"A } \lor \text{ B" or "A } \cup \text{ B" or "A or B".}$$

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 this 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 $x_i$, “$X$ 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 $X \in \{\text{heads, tails}\}$. If the coin comes up heads, then $X = \text{ heads}$. If the coin comes up tails, then $X = \text{ tails}$. For a fair coin we have

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

\(^2\)One can also define continuous variable (e.g., that take values continuously between 0 and 1), where sums are replaced by variables.
Note that these two probabilities sum to 1.

If we have two random variables $X \in \{x_1, \ldots, x_k\}$ and $Y \in \{y_1, \ldots, y_\ell\}$, 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 $X, Y \in \{\text{heads, tails}\}$ 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,

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

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

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 $E[X]$, is the weighted sum

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

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

18
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

\[
E[X + Y] = E[X] + E[Y].
\]

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:

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

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

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

19
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

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

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,

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

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

$$E_X\left[ E_Y[f(X,Y) | X] \right] = \sum_x E_Y[f(X,Y) | X = x] P[X = x] = \sum_x \sum_y f(x,y) P[Y = y | X = x] P[X = x]. \quad (1.5)$$

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

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

(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. Randomized searching and sorting

1.3 Randomized approximations for SAT

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.

$$\begin{array}{c|cc}
P[\cdot, \cdot] & E_2 & \bar{E}_2 \\
\hline
\bar{E}_1 & 49/64 & 7/64 \\
\bar{E}_1 & 7/64 & 1/64 \\
\end{array}$$
1. Randomized searching and sorting

1.3. Randomized approximations for SAT

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, 

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

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 

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

Altogether we obtain

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

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) \(7/4\).

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

\[^{3}2 \cdot (49/64) + 7/64 + 7/64 = 7/4.\]
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, $E[Z] = (7/8)m$. 

23
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

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

Rearranging, we have

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

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 \( \mathbb{E}[Z | x_1 = t] \) and \( \mathbb{E}[Z | x_1 = f] \). Fortunately this is easy to do. Consider the case \( x_1 = t \). By linearity of expectation we have

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

For a particular \( Y_i \), we have

1. \( \mathbb{E}[Y_i | x_1 = t] = 1 \) if \( x_i \) appears (unnegated) in the \( i \)th clause, and
2. \( \mathbb{E}[Y_i | x_1 = t] = 1 - 2^{-k_i} \) otherwise, where \( k_i \) is the number of undecided variables remaining.\(^4\)

Thus we can calculate \( \mathbb{E}[Z | x_1 = t] \) and similarly \( \mathbb{E}[Z | x_1 = f] \) 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

\[
\mathbb{E}[Z | x_1 = a_1, \ldots, x_k = a_k] \geq (7/8)m.
\]

\(^4\)For 3-SAT, \( k_i = 2 \) if \( \overline{x}_i \in C_i \), and \( k_i = 3 \) otherwise.
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.

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

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

26
Proof. Since $X$ is nonnegative we have

$$E[X] = E[X \mid X \geq \alpha] P[X \geq \alpha] + E[X \mid X < \alpha] P[X < \alpha] \geq \alpha P[X \geq \alpha]$$

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

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

$$P[D_e \geq 32 \ln n] \leq 1/n^3. \quad (1.6)$$

Assuming (1.6) holds for any $e$, we then have

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

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 $E[X_i]$ for each $i$ (and eventually show that $E[X_i] = 1/\text{poly}(n)$ for $i = O(\log n)$). As mentioned above, $X_0 = n$. Consider $X_1$. An exact estimate for $E[X_1]$ 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

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

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

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

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

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

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

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

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

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

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

as claimed in eq. (1.6). This completes the proof.

1.5 Additional notes and materials

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

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

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.
Spring 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 in [MR95, Chapter 1].

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 or disprove 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

$$\mathbb{E}_X[\mathbb{E}_Y[f(X, Y) | X]] = \mathbb{E}_Y[\mathbb{E}_X[f(X, Y) | Y]].$$

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, “$1 + x ≈ e^x$”.

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

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

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

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.

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

Exercise 1.11. For $k ∈ \mathbb{N}$, suppose you repeatedly flip a coin that is heads with fixed probability $p ∈ (0, 1)$.

1. What is the expected number of coin flips until you obtain one heads?\footnote{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 $p = 1/2$.} 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.

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 $O(1)$.

   \[6\text{Hint: Exercise 1.11.}\]
(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 $O(n)$.

**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 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 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 $1/2$. \(^7\)

2. Use the preceding result to show that if $AB \neq C$, then with probability at least $1/2$, $ABx \neq Cx$. \(^8\)

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 $AB \neq C$.

\(^7\)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.
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”.

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.

\footnote{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.}
Chapter 2

Hashing and heavy hitters

2.1 Introduction

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

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

Google currently serves billions\(^1\) of queries a day. Given the sheer magnitude of Google’s search traffic, and the diversity of search queries, it is not obvious how to

\(^1\)Maybe 7 billion? See https://www.internetlivestats.com/google-search-statistics/
identify the most popular search queries. Certainly one cannot simply have a tally for each search term, since there are too many search terms out there to be stored. More generally, it is prohibitively expensive to maintain any data structure proportional to the input size. Somehow we need an approach that takes sublinear space.

Streaming models. We study the heavy hitter problem in the streaming model of computation. In the streaming model, the input is a sequence of items presented to the algorithm one at a time. We assume the algorithm has 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.

The heavy hitters problem. We formalize the heavy hitters problem as follows. We have $m$ elements in a stream $e_1, \ldots, e_m$, where each element is from some large universe $[n] = \{1, \ldots, n\}$. Elements can repeat. The absolute frequency of an element $e$, denoted $f_e$, is the number of times the element appears in the stream. The relative frequency, denoted $p_e$, is the fraction of the stream that the element appears in. In a stream of $m$ elements, the relative frequency of an element is the 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. 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 tractable and useful. We may have to relinquish exactness and consider 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 randomized algorithms that have some small – analyzed and controlled – probability of failure.

\(^2\)One can formalize this impossibility as follows. There are $\binom{m+n-1}{n-1}$ ways to make a frequency
2. Hashing and heavy hitters

2.1. Introduction

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 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 $\lfloor 1/\epsilon \rfloor$-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 $m$. (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 $(m+n-1)_{n-1}$ outcomes, so we must have $2^k \geq (m+n-1)_{n-1}$, hence $k \geq \log((m+n-1)_{n-1})$. Here log denotes $\log_2$. We also have

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

and also

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

Thus

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

and

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

Slightly better low bounds can be obtained via Stirling’s approximation, which is also related to entropy.
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 $h : [n] \to [k]$ where the values $h(i)$ are (in a qualified sense) randomly distributed through $[k]$. A collision is a pair of distinct indices $i_1 \neq i_2 \in [n]$ such that $h(i_1) = h(i_2)$. In most applications, $n$ is much (much, much) larger than $k$. In this case, there are necessarily many “collisions” (see exercise 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 $h : [n] \to [k]$ is a uniformly random function $h : [n] \to [k]$. That is, each $h(i)$ is drawn from $[k]$ independently and uniformly at random.

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

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

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

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

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 \((\epsilon m)\)-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 \(1/\epsilon\) 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

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

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

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

Here (c) plugs in the inequality obtained in (2.1). (d) applies Markov’s inequality, where we note that $A[h(e)] - f_e \geq 0$. \qed
# 2. Hashing and heavy hitters

## 2.4. Amplification

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 $1/2 = .5$. With two coin tosses, the probability that both come up tails is still $1/4 = .25$. But with 100 coin tosses, the probability that all 100 coin tosses come up tails is $1/2^{100} \approx .0000000000000000000000000000007886...$.

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(\epsilon)`. 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

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

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

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

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 \([n]\), \( \text{count-min}(\epsilon, 1/n^2) \) has the following guarantee at any fixed point in the stream.

Suppose \( m \) elements have been presented in the stream. With probability at least \( 1 - 1/n \), \( \text{count-min}(\epsilon, 1/n^2) \) 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 \( \leq 1/n \). \qed

**Remark 2.7.** More precisely, the space usage of \( \text{count-min}(\epsilon, 1/n^2) \) 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*. 
2. Hashing and heavy hitters

2.6. Takeaways

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 [Che14b; Nel20].

**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. Hashing and heavy hitters

2.8 Exercises

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

2.8 Exercises

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

1. Prove that the number of collisions is

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

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

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

To recall the construction, we randomly construct a function $h : [n] \rightarrow [k]$ 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

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

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 $b \in \{0, \ldots, p-1\}$.

• 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 $\{(c_1, c_2) \in \{0, \ldots, p-1\} : c_1 \neq c_2\}$.

\[3\] 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 $bc = a$. 
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
\[
\sum_{\substack{c_2 \in \{1, \ldots, p\} \\ c_2 \neq c_1}} P[ax_1 + b = c_1, ax_2 + b = c_2] \leq \frac{1}{pk}.
\]
- The LHS represents \( P[ax_1 + b = c_1 \mod p \text{ and } h(x_2) = h(x_1)] \).

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

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 \( \leq 1/poly(n) \) 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.

---

4Here we note that for \( x_1 \neq x_2 \), \( ax_1 + b = ax_2 + b_2 \mod p \) iff \( a = 0 \).
5There was a typo where previously the RHS said “1/n” instead of “1/pk”.
6Hint: You may want to show that the number of values \( c_2 \in [p] \) such that \( c_1 = c_2 \mod k \) is \( \leq \frac{p-1}{k} \).
7Here the elements are integers from \( [n] = \{1, \ldots, n\} \), where \( n \) is known, and \( \epsilon \in (0, 1) \) is an input parameter.
8To 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.
9You 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 \( \delta \).
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.\footnote{Hint: It might be helpful to think about the special case of $S = 0$.}\footnote{Hint: $k$ was chosen to be greater than the maximum number of heavy hitters. Why?}\footnote{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.}

**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:
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 \( \{e_1, \ldots, e_m\} \).\(^{13}\)

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.\(^{14}\)

---

\(^{13}\)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. \texttt{set}(k, v): Associate the value \( v \) with the key \( k \).

2. \texttt{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 \texttt{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 \texttt{Map} interface in Java and for \texttt{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.
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.

A deterministic way to implement dictionaries is via search trees. If the keys are comparable (such as numbers, or strings in alphabetical order), then search trees can
organize the data in sorted order in a tree-like data structure. With a well-designed search tree, searching for a key has roughly the performance of a binary search over a sorted array: $O(\log n)$ time per \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 Sleator and Tarjan [ST85b] 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 [Mor68]), these algorithms can be compact and efficient. Now, as clever as these data structures are, they suffer some drawbacks compared to arrays. The $O(\log n)$ query time for search trees is a bit higher than the $O(1)$ time of arrays\footnote{Sometimes, log factors matter.}. They are more complicated to implement, and require a lot of \texttt{pointer chasing}, which leads to many cache misses on the CPU.\footnote{This last point can be helped to some extent by \textit{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 $h : [U] \to [m]$. Ideally, each key-value pair $(k_i, v_i)$ is stored in the slot $A[h(k_i)]$. The remaining question is what to do when keys collide, i.e., when $h(k') = h(k'')$ for two distinct keys $k'$ and $k''$. There are various ways, 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 [KLLPLL97]. Distributed dictionaries are particularly useful for caching on the web.
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 $A[1, ..., m]$, 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

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

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

### 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 $A[1, ..., m]$ and a hash function $h : \{1, ..., U\} \to \{1, ..., m\}$. 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 [Knu63]. 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, 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.

---

4He 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.

5See for example this interview: https://www.youtube.com/watch?v=Wp7GAKLSGnI.
Post-Knuth, there remained a question of how much independence was required to get 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, Schmidt and Siegel [SS89; SS90] showed that \( O(\log n) \)-wise independence sufficed\(^6\). Then, in 2007, Pagh, Pagh, and Ruzic [PPR09] showed that (just!) 5-wise independence sufficed. This was dramatic progress for arguably the oldest problem in algorithm design. Soon after, [PT16] showed that 4-wise independence was not enough. So the answer is 5!

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

### 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.

**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

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

---

\(^6\) Alan Siegel taught me algorithms.
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 $\Pr[X_i = 1] = p$ for each $i$. Let $\mu = pn$ be the expected sum, and suppose $\mu \geq 1$. Then for all $\beta > \mu$,

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

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

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

(3.1)

Lemma 3.5 says that

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

(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”.

---

56
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

\[
\mathbb{E}[\text{running time}] \leq \mathbb{E}[|R|] = \sum_{\ell=1}^{n} \ell \mathbb{P}[|R| = \ell] \leq \sum_{k=1}^{\lceil \log n \rceil} 2^k \mathbb{P}[2^{k-1} < |R| \leq 2^k].
\] (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 \( > 2^{k-1} \), then at least \( 2^{k-1} \) items other than \( x \) hash to \( R \). Thus for each \( k \), we have

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

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

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

We have

\[
\mu = \frac{|I_k|n}{m} \leq 2^{k-2},
\]

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

\[
\mathbb{P}\left[ \left( \# \text{ other items hashing into } I_k \right) > 2^{k-1} \right] \leq \frac{4 \max\{\mu, \mu^2\}}{(2^{k-1} - \mu)^4} \leq \frac{4(2^{k-2})^2}{(2^{k-2})^4} \leq \frac{1}{22^{k-6}}.
\]
Because (b) is by lemma 3.8. Plugging back into RHS(3.3) above, we have

$$E[\text{running time}] \leq \text{RHS}(3.3) \leq \sum_{k=1}^{\lceil \log n \rceil} 2^k \cdot \frac{1}{2^{2k_6}} = 2^6 \sum_{k=1}^{\lceil \log n \rceil} \frac{1}{2^k} \leq 2^6.$$

A constant! \hfill \Box

### 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

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

Before proving lemma 3.7, let us give a simple example where $k$-wise independence matters. Let $X_1, \ldots, X_k \in \{0, 1\}$ 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 come 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
\[ E[X_1 X_2 \cdots X_k] \]
\[ \overset{(a)}{=} \sum_{y_1, y_2, \ldots, y_k} y_1 y_2 \cdots y_k P[X_1 = y_1, X_2 = y_2, \ldots, X_k = y_k] \]
\[ \overset{(b)}{=} \sum_{y_1, y_2, \ldots, y_k} y_1 y_2 \cdots y_k P[X_1 = y_1] P[X_2 = y_2] \cdots P[X_k = y_k] \]
\[ = \left( \sum_{y_1} y_1 P[X_1 = y_1] \right) \left( \sum_{y_2} y_2 P[X_2 = y_2] \right) \cdots \left( \sum_{y_k} y_k P[X_k = y_k] \right) \]
\[ \overset{(c)}{=} E[X_1] E[X_2] \cdots E[X_k]. \]

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

The key step is (a), where we raise both sides to the fourth power. (b) is by Markov’s inequality. We claim that
\[ E\left[ \left( \sum_{i=1}^{n} X_i - \mu \right)^4 \right] \leq \mu + 3\mu^2, \]
\(^7\)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, \(P[X_1 = y_1, \ldots, X_k = y_k]\).
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
\[ \sum_{i=1}^{n} (X_i - p)^4 + \binom{4}{2} \sum_{i<j} (X_i - p)^2 (X_j - p)^2 + \text{(monomials w/ some)} \]
\( (X_i - p) \text{ w/ degree 1} \). (3.4)

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

1. For each \( i \), we have
\[ E[(X_i - p)^4] = p(1-p)^3 + (1-p)p^3 \leq p(1-p). \]

2. For each \( i \neq j \), we have
\[ E[(X_i - p)^2(X_j - p)^2] = E[(X_i - p)^2] E[(X_j - p)^2] \leq p^2(1-p)^2. \]

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,
\[ E[(X_1 - p_1)^3(X_2 - p_2)] = E[(X_1 - p_1)^3] E[X_2 - p_2] \leq 0, \]
where (e) is by pairwise independence, and (f) is because \( E[X_2 - p_2] = 0 \).

Plugging back in above, we have
\[ E \left[ \left( \sum_{i=1}^{n} X_i - \mu \right)^4 \right] = np(1-p) + \binom{n}{2} \left( \frac{4}{2} \right) (p^2(1-p)^2) \leq np + 3(np)^2, \]
as desired. This completes the proof. \( \square \)

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

- Dictionary data structures provide everyday motivation for studying randomization, where hash tables offer simpler and better performance (in expectation) than search trees.

- There are different ways to implement hash tables and they mostly differ in how they handle collisions.

- Chaining uses linked lists to handle collisions. It has reasonable performance in expectation for universal hash functions, and stronger guarantees when the hash function is more independent.

- Linear probing is perhaps the easiest hash table to implement, and scanning an array is hardware-friendly. It had been observed to perform well in practice long before it had been properly analyzed.

- The analysis of linear probing cleverly uses canonical intervals (doubling in size) to limit the number of “bad events” we have to avoid, to roughly \( \log n \) (per key).

- It turns out that 5-wise independence is sufficient for linear probing to have \( O(1) \) running time in expectation. Interestingly, 4-wise independence is not enough.

3.6 Additional notes and materials

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

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

**Spring 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 \( h : [n] \rightarrow [\ell] \) 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 \( O(\log(n)/\log \log n) \).

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 \( \geq 1/2 \).

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\(^9\)
   \[
   \sum_{i=1}^{n} k_i^2 \leq n + O(\text{total # of collisions (w/r/t } h_0)).
   \]

---

\(^8\)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.

\(^9\)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.
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 \textit{static settings}, when the set of keys is fixed.
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 \(\sum_{e \in C} c(e)\) over all cuts \(C \subset E\). (4.1)

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

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

4.1. Minimum cut

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

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

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

Kent Quanrud
Fall 2022

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

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

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

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

66
let $c/e : E/e \to \mathbb{R}_{>0}$ denote the corresponding capacities. Figure 4.3 sketches a few iterations of the algorithm applied to a barbell graph.

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

$$\sum_{e \in C} c(e) \overset{\text{def}}{=} \sum_{e \in C} c(e).$$
Lemma 4.1. Let $C^*$ 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 $P[e \in C^*] \leq \frac{2}{n}$.

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

Lemma 4.4. Let $C^*$ be a minimum cut. With probability $\geq \frac{1}{\binom{n}{2}}$, 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, $P[E_0] = 1$, and we want to show that $P[E_{n-2}] \geq \frac{1}{\binom{n}{2}}$. By lemma 4.3, we have

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

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

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

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
4. Sampling edges

4.2 Amplification by branching

\[ \text{branching-contractions}(G = (V, E), c) \]

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. \( C_1 \leftarrow \text{branching-contractions}(G, c) \).
4. \( C_2 \leftarrow \text{branching-contractions}(G, c) \).
5. Uncontract and return the minimum of \( C_1 \) and \( C_2 \).

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

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

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

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

Karger and Stein [KS96] proposed randomly 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 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 these 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 are interested in

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

$$= \frac{\prod_{i=3}^{n} \frac{i-2}{i}}{\prod_{i=3}^{k} \frac{i-2}{i}} = \left(\frac{k}{2}\right) = \frac{k(k-1)}{n(n-1)}.$$ 

by (a) lemma 4.3. \qed

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

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

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

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

$$\varphi^i(n) = \left(\frac{1}{\sqrt{2}}\right)^i n + \sum_{j=0}^{i} \left(\frac{1}{\sqrt{2}}\right)^i c \leq \left(\frac{1}{\sqrt{2}}\right)^i n + \frac{c}{\sqrt{2} - 1}.$$
4. Sampling edges
4.2. Amplification by branching

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

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

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

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

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

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

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

The first three values are

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

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

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

\[\square\]
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 and Ibaraki [NI92] 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 \( F \).
2. If \( s,t \in V \) have connectivity \( \ell \geq k \) in \( G \), then \( s \) and \( t \) have connectivity \( \geq k \) in \( G \).

Nagamochi and Ibaraki [NI92] 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 and Ibaraki [NI92]). 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.

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 \( e \in F_k \).

### 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 \(\leq O\left(\log^2(n)/i\epsilon^2\right)\). 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

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

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

**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,
\[
\mathbb{E} \left[ \sum_{e \in C \cap E'} w(e) \right] \overset{(a)}{=} \sum_{e \in C} \mathbb{P}[e \in E'] \mathbb{E}[w(e) | e \in E'] = \sum_{e \in C} p_e \cdot \frac{1}{p_e} = |C|.
\]
Here (a) is by linearity of expectation.

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 \( \beta > 0 \),
\[
\mathbb{P} \left[ \sum_{e \in S \cap E'} w(e) \geq (1 + \epsilon) |S| + \beta \right] \leq e^{-c \log^2(n) \beta / \epsilon |S|}
\]
and
\[
\mathbb{P} \left[ \sum_{e \in S \cap E'} w(e) \leq (1 - \epsilon) |S| - \beta \right] \leq e^{-c \log^2(n) \beta / \epsilon |S|}.
\]

**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
\[
w(e) = \frac{1}{p_e} \leq \frac{e^2 c k}{\log^2(n)} = \left( \frac{e^2}{c \log^2(n)} \right) |C|.
\]
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 $\left(\frac{\epsilon^2}{c \log^2(n)}\right)$-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 $\beta = \frac{\epsilon|S|}{c}$, 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 $n$ 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 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

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

and

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

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 distinct $n^{2\alpha}$ 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 then simply assuming where $2^n$ 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 $e \in H_k = F_{2^{k-1}} \cup \cdots \cup F_{2^k}$ 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

$$A_\ell = \{H_k \cap C : 2^{k-1} \leq |C| < 2^{k+1}\}.$$

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

$$|A_\ell| \leq n^{c_0 \ell} \tag{4.4}$$

for some constant $c_0 > 0$. For each section $A = H_k \cap C$ where $C \in A_\ell$, we have

$$(1 - \epsilon)|A| - \frac{\epsilon}{\log n}|C| \leq \sum_{e \in A \cap E'} w(e) \leq (1 + \epsilon)|A| + \frac{\epsilon}{\log n}|C| \tag{4.5}$$

with probability of error

$$\leq e^{-c\log(n)|C|} = n^{-c_1|C|/2^k} \leq n^{-c_1 \ell/2}$$

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

$$\leq n^{-(c_0+2)\ell} \tag{4.6}.$$

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

$$|A_\ell| \cdot n^{-(c_0+2)\ell} \leq n^{-2\ell+1}.$$

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

79
4. Sampling edges

4.4. Sparsification

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.

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 [Fra11].

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 $\leq \alpha K$.

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

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

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

4.4.3 Weighted, Sparser, Stronger... and Deterministic

Benczúr and Karger [BK15] 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. [BK15] was actually the first such sparsification result, and precedes the simpler algorithm described above, which is from Fung, Hariharan, Harvey, and Panigrahi [FHHP19]. We point out two generalizations of cut sparsifiers, which we plan to return to later. Spielman and Srivastava [SS11] 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. Batson, Spielman, and Srivastava [BSS12] 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 [MSS13].

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 \(s \to t\) flow. Recall the Ford-Fulkerson algorithm, a staple of introductory algorithms classes.

\[\text{Repeatedly find a path from } s \text{ to } t \text{ in the residual graph, and route one unit of flow.}\]

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 \(O(m + n^2 \log^2(n)/\epsilon^2)\) 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 and Levine [KL15] 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.

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) $O\left(\frac{1}{n} n \log^3 n \right)$ 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 the following algorithm.

In the $i$th iteration, sample $O\left(\frac{\lambda n}{(\lambda + 1 - i)} \text{polylog}(n) \right)$ 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.
4. Sampling edges

4.5. Randomized Ford-Fulkerson

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 [KL15].

Theorem 4.24 ([KL15]). 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 $O((m + n\lambda) \text{polylog}(n))$ randomized time with high probability.

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 $s \rightarrow t$ path. Assuming lemma 4.23 is true, on the $i$th of $\lambda$ iterations, we will find an $s \rightarrow t$ path with high probability as soon as we sample at least

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

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.\(^3\)

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

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

\(^2\)Actually, the algorithm of Karger and Levine [KL15] is slightly different, because it samples edges in proportion to sampling probabilities given by Benczúr and Karger [BK15]. The high level ideas are the same, although the log factors are improved by using the [BK15] sampling probabilities.

\(^3\)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 $\theta$. 

84
We also have $m\, \text{polylog}(n)$ overhead from computing the sampling probabilities $p_e$ initially. This gives the claimed running time. □

4.5. Analysis

Let $f$ be an $s \to t$ flow. Fix an undirected $\{s,t\}$-cut of the form $C = \delta(S)$, where $S \subseteq V$, $s \in S$, 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.

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

**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 $s \to t$ 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 $\delta$.

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 $s \to t$ 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. □

**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.

---

4The author encourages student readers to verify this in full detail.
Suppose we sample \( O(\alpha n \log^3(n)) \) 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, 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.

**Lemma 4.27.** Let \( \beta > 1 \), and suppose we sample \( O(\beta n \log^3(n)) \) 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 \( |C_f| \geq \alpha |C|/2 \).

---

5As 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 \( \beta > 0 \),

\[
P[X_1 + \cdots + X_n \leq (1 - \epsilon) \mathbb{E}[X_1 + \cdots + X_n] - \beta] \leq e^{-\epsilon^2 \beta}.
\]

6Again, the student reader should verify the calculations for themselves.
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 $s \rightarrow t$ path.

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

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

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 $s \rightarrow t$ path. 

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.\(^7\)

\(^7\)You might find it helpful to focus on the case $k = 3$ and then generalize afterwards. Although we ask you to work out the dependency on $k$, conceptually, it might help to think of $k$ as being relatively small compared to $n$. 87
1. Briefly describe how to modify the random-contractions to return a $k$-cut.\(^8\)

2. Analyze the probability that your modified algorithm returns a minimum $k$-cut.\(^9\)

3. Describe and analyze an algorithm, using your modified random-contractions as a subroutine, that computes a minimum $k$-cut with high probability in $O\left(n^{c_1 k \log^2 n}\right)$ time for constants $c_1$ and $c_2$. (We leave it to you to identify these constants; as usual, the faster the running time, the better.)

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

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

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

2. Show the number of 2-approximate minimum cuts is at most $O(n^4)$.

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

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

2. Show the number of $\alpha$-approximate minimum cuts is at most $n^{O(\alpha)}$.

**Exercise 4.5.** Let $G = (V, E)$ be a simple Eulerian directed graph\(^{10}\).

---

\(^8\)Your algorithm design may be informed by your calculations in part 2.  
\(^9\)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.  
\(^{10}\)A directed graph with Eulerian if every vertex has the same in-degree as out-degree.
4. Sampling edges

4.7. Exercises

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.
Chapter 5

Random Sums and Graphs

5.1 Random sums

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

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

Suppose you were told that at most 25 coin tosses came up heads. Should you be surprised? On one hand, 25 is half of the expected amount. On the other hand, the claim is not that there was exactly 25 heads, but at most 25 heads. There could be 25, 24, 23, etc., down to 0. 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 $1.364 \times 10^{-10}$. 

90
5. Random Sums and Graphs

5.1. Random sums

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 [Wikb].)

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 \( \mathbb{E}[B] = np \). The following lemma bounds the probability of \( B \) being a multiplicative factor smaller than its mean, \( np \). 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

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

**Proof.** We have

\[
\mathbb{P}[B \leq (1 - \epsilon)np] = \mathbb{P}[e^{-\epsilon B} \geq e^{-(1-\epsilon)np}] \leq \frac{\mathbb{E}[e^{-\epsilon B}]}{e^{(1-\epsilon)pn}} = e^{(1-\epsilon)pn} \mathbb{E}[e^{-\epsilon B}] \tag{5.1}
\]

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

\[
\mathbb{E}[e^{-\epsilon B}] = \mathbb{E}[e^{-\epsilon X_1 + \cdots + \epsilon X_n}] \overset{(b)}{=} \mathbb{E}[e^{-\epsilon X_1}] \mathbb{E}[e^{-\epsilon X_2}] \cdots \mathbb{E}[e^{-\epsilon X_n}],
\]

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

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

$$E[e^{-\epsilon B}] = E[e^{-\epsilon X_1}] E[e^{-\epsilon X_2}] \ldots E[e^{-\epsilon X_n}] \leq e^{-(\epsilon - \epsilon^2/2)pn}, \quad (5.2)$$

Putting everything together, we have

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

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

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

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

### 5.2 Random graphs

Paul Erdős, inspired by Ramsey [Ram30] before him, had a series of work analyzing random graphs, producing a large body of results that can mostly be grouped into

$\footnote{To see that}$

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

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 $G(n,.01)$ [Von].](image)
two broad categories. First, he designed elaborate randomized constructions of graphs and showed that with nonzero probability, they can possess certain counterintuitive, seemingly impossible properties. This general approach is now called Ramsey theory. Second, he showed that for natural random graph models, these graphs – however random – tend to be extremely consistent about certain properties. Today we will study the \( G(n, p) \) random graph, sometimes called Erdös-Rényi graphs based on work by Erdös and Rényi [ER59; ER60]. 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, Erdös and Rényi showed that these properties can vary dramatically with very small changes in \( p \). Consider the following theorem.

**Theorem 5.3.** Consider a random graph \( G \sim G(n, p) \) 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 [Bol98, Chapter 7] for further details and other results in this area.

We present the proof for \( c > 1 \). The second case of \( c < 1 \) is left to the reader as exercise 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 [Kar90] 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 [Kar90, Lemma 1] (with minor changes in notation) whose proof is very elegant.

...To see that the last two random variables are identically distributed, note that the probability spaces \( B(n, p) \) and \( D(n, p) \) differ in only one respect: a digraph \( G \) drawn from \( B(n, p) \), are \((u, v)\) is present if and only if arc \((v, u)\) is present, while, in a digraph \( D \) drawn from \( D(n, p) \), then the event that \((v, u)\) is present is independent of the event that \((u, v)\) is present. Thus no experiment based on checking for the presence or absence of arcs can distinguish between the two probability spaces unless it checks both an arc and its reversal. But any standard sequential algorithm, such as breadth-first search or depth-first search, for building a search tree containing exactly the vertices reachable from vertex 1, checks for the presence of arc \((u, v)\) only if vertex \( u \) is in the search tree and \( v \) is not; thus it never checks both an arc and its reversal, and accordingly cannot distinguish \( B(n, p) \) from \( D(n, p) \).

To summarize the excerpt, standard search algorithms for reachability do not distinguish \( B(n, p) \) and \( D(n, p) \) anyway, so the number of reachable vertices is identically distributed. \( \square \)

### 5.3 A gap in component size

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

**Lemma 5.4.** There is a universal constant \( C > 0 \), such that for all \( \epsilon \in (0, 1) \), and for all \( n > 0 \) sufficiently large, and \( p = (1 + \epsilon)/n \), we have the following. For a random graph \( G(n, p) \), 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 \( \text{nil} \) 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 \( C(v) = B_{i-1} = A_{i-1} \).

The process terminates when \( B_i = A_i \). But since \( B_i \subseteq A_i \) and \( |B_i| = i \), this is precisely when \( |A_i| = i \). As long as \( |A_i| \neq i \), \( A_{i+1} \) 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 \( v_i \).
1. Initially set $A_0 = \{v\}$.

2. For each $i \in \mathbb{N}$, let $S$ sample each vertex in $V \setminus A_{i-1}$ independently with probability $p$ and set $A_i = A_{i-1} \cup S$.

3. Let $i$ be the first index such that $|A_i| = i$, and return $C(v) = A_i$.

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

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

**Proof.** We have 

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

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

$$\mathbb{E}[|A_i|] = 1 + (1 - (1 - p)^i)(n - 1) \geq (1 - \epsilon/4)ipn \geq (1 + \epsilon/2)i.$$ 

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

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

**Proof.** We have 

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

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 

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

By the union bound, we have 

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

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.
5.4 Galton-Watson process with general branching factors

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

**Lemma 5.5.** Let $p = (1 + \epsilon)/n$ for $\epsilon > 0$. Let $v \in V$ be a vertex. For all $3 \leq h \leq \epsilon n$, 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

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

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 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.

100
5.6 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.)

²It may be helpful to understand the proof of the gap theorem (lemma 5.4) in section 5.3.
Chapter 6

Randomized Rounding

Many of the problems we are interested in are inherently discrete, and unfortunately many discrete problems are NP-Hard. One general class of 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

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

(6.1)

where \( A \in \mathbb{R}_{\geq 0}^{m \times n} \), \( b \in \mathbb{R}_{> 0}^{m} \), and \( c \in \mathbb{R}_{> 0}^{n} \). Here, each variable \( x_j \in \mathbb{Z}_{\geq 0} \) models a binary decision. Each \( c_j \) can be interpreted as the cost of taking \( x_j = 1 \). For \( i \in [m] \), the constraint

\[
\sum_{j=1}^{n} A_{ij} x_j \geq b_i \text{ for } i \in [m]
\]

is a covering constraint, saying we must set \( x_j = 1 \) for enough variables \( x_j \) so that the sum of \( A_{ij} \) over these variables is at least \( b_i \).

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

$$\text{maximize } \sum_{j=1}^{n} b_j y_j$$

over $y \in \mathbb{Z}_{\geq 0}^n$

s.t. $\sum_{j=1}^{n} A_{ij} y_j \leq c_j$ 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. 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.

$$\text{minimize } \sum_{j=1}^{n} c_j x_j$$

over $x \in \mathbb{R}_{\geq 0}^n$

s.t. $\sum_{j=1}^{n} A_{ij} x_j \geq b_i$ 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.

103
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., [Vaz01; WS11]). 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 “\( \bar{x}_j \in C_i \)” indicate that \( \bar{x}_j \) appears in \( C_i \).

Consider now the following integer program.

\[
\text{maximize } \sum_{i=1}^{m} z_i \text{ over } y \in \{0, 1\}^n, z \in \{0, 1\}^m \\
\text{s.t. } \sum_{j: x_j \in C_i} y_j + \sum_{j: \bar{x}_j \in C_i} (1 - y_j) \geq z_i \text{ for all clauses } C_i.
\]

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 \( \bar{x}_j \in C_i \).

**A linear program for SAT.** As discussed, while we cannot solve integer programs in polynomial time, we can solve linear programs. We relax the integer program above to a linear program by now allowing each variable \( y_j \) or \( z_i \) to lie anywhere in the interval \([0, 1]\).

\[
\text{maximize } \sum_{i=1}^{m} z_i \text{ over } y_1, \ldots, y_n, z_1, \ldots, y_m \in \mathbb{R} \\
\text{s.t. } \sum_{j: x_j \in C_i} y_j + \sum_{j: \bar{x}_j \in C_i} (1 - y_j) \geq z_i \text{ for all clauses } C_i \\
0 \leq z_i \leq 1 \text{ for all } i = 1, \ldots, m \\
0 \leq y_j \leq 1 \text{ for all } j = 1, \ldots, n
\]

Let \( \text{OPT}_{lp} \) denote the optimum value of the LP. Since the LP is a relaxation of the IP above it, we have \( \text{OPT}_{lp} \geq \text{OPT} \).
Randomized Rounding. 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 \( \text{OPT}_{\text{LP}} = \sum_i z_i \) is very good. Our goal now is to convert the \( y_j \)'s into discrete decisions while keeping the objective value as close to \( \text{OPT}_{\text{LP}} \) as possible.

The basic question is: how do we interpret a fractional value such as \( y_1 = 0.5 \)? 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

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

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

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

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 \( t \in [0, 1] \).
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 $(1 - 1/e) \text{OPT}_{\text{LP}} \geq (1 - 1/e) \text{OPT}$ 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 $\geq (3/4)z_i$.

**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

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

(a) is by AM-GM. (b) is by convexity of $f(x) = \left(1 - \frac{x}{2}\right)^2$. \hfill \qed

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></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

\[
\text{find the minimum number of sets } S_{i_1}, \ldots, S_{i_k} \text{ such that } S_{i_1} \cup \cdots \cup S_{i_k} = [m].
\]

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.
6. Randomized Rounding

6.2. Set cover

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 \( F \subseteq \{S_1, \ldots, S_m\} \) by 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 \( F \) doesn’t cover \( i \). \( F \) does not cover \( i \) iff it fails to sample any of the sets covering \( i \), hence

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

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 \( 1 - \frac{1}{m^2} \).

**Proof.** If \( x_j \geq \frac{1}{\alpha} \) for any set \( S_j \) covering \( i \), then \( i \) is covered deterministically. Otherwise, by similar calculations as before, we have

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

**Lemma 6.6.** \( F \) is a set cover with probability at least \( 1 - \frac{1}{m} \).

**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 \( 1 - \frac{1}{m} \). 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

\[
\text{minimize } \langle c, x \rangle \text{ over } x \in \mathbb{Z}_{\geq 0}^n \text{ s.t. } Ax \geq b.
\]
where \( c \in \mathbb{R}^n_{>0}, A \in \mathbb{R}^{m \times n}_{\geq 0}, \) and \( b \in \mathbb{R}^m_{>0}. \) The matrix notation expands out to the following.

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

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 a \( O(\log m) \) factor, and then randomly round each coordinate of \( x, \) independently, to integer values.

1. Let \( \alpha = 8 \log(m). \)
2. For each \( j \in [n], \) independently, let

\[
\begin{align*}
z_j &= \begin{cases} 
\lceil \alpha x_j \rceil & \text{with probability } \alpha x_j - \lfloor \alpha x_j \rfloor, \\
\lfloor \alpha x_j \rfloor & \text{otherwise.}
\end{cases}
\end{align*}
\]

The algorithm is very similar to set cover. However, the coefficients \( A_{ij} \) make the analysis more difficult. For set cover, we were able to give an exact formula for the probability that a point \( i \) is uncovered. (Equation (6.2).) Here it is not so simple, and we instead appeal to the multiplicative (or relative) Chernoff bound introduced last chapter. The proof is deferred to section 6.A.

**Theorem 6.8.** Let \( X_1, \ldots, X_n \in [0, 1] \) be independent random variables, and \( \epsilon \in [0, 1]. \)

- For \( \mu \geq \mathbb{E}[X_1 + \cdots + X_n], \)

\[
P[X_1 + \cdots + X_n \geq (1 + \epsilon)\mu] \leq e^{-c^2 \mu/3}.
\]

- For \( \mu \leq \mathbb{E}[X_1 + \cdots + X_n], \)

\[
P[X_1 + \cdots + X_n \leq (1 - \epsilon)\mu] \leq e^{-c^2 \mu/2}.
\]

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

6.3. Covering integer programs

Lemma 6.9. The $i$th constraint is satisfied with probability at least $1 - 1/m^2$.

Proof. By scaling, we may assume that $b_i \leq 1$, hence $A_{ij} \leq 1$ for all $j$. We may assume that $\sum_j A_{ij} \lfloor \alpha x_j \rfloor < 1$, since otherwise the constraint is satisfied deterministically.

For each coordinate $j$, let $X_j = A_{ij}(z_j - \lfloor \alpha x_j \rfloor)$ for each $j$. Let $\mu = \mathbb{E}\left[ \sum_{j=1}^{n} X_j \right]$. 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

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

We have

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

By the Chernoff inequality, we have

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

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

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

as desired. \qed
Theorem 6.10. With constant probability, the randomized rounding algorithm returns a feasible solution of cost at most $O(\ln(m)) \cdot \text{OPT}_{LP}$.

Proof. By Markov’s inequality, we have

$$\sum_j c_j z_j \leq 2 \mathbb{E} \left[ \sum_j c_j z_j \right] = 2 \alpha \cdot \text{OPT}_{LP}$$

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 $2 \cdot \text{OPT}_{LP}$ 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. \qed

6.4 Additional notes and materials

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

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

6.5 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.\(^3\)

\(^3\)You may first want to design and analyze a deterministic $(1 - 1/e)$-approximation algorithm for max-SAT.
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 `round-and-fix` for which some pseudocode is given below. `round-and-fix` is similar to randomized rounding and has two stages. The first stage solves the LP and then rounds the solution scaled up by some factor $\alpha \geq 1$. It is possible that some of the elements $i \in [m]$ may not be covered. In the second stage, we fix each uncovered element by (deterministically) taking the cheapest set that covers it.

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

1. let $x \in [0,1]^n$ solve the set cover LP
2. let $F \subseteq \{S_1, \ldots, S_n\}$ sample each set $S_i$ independently with probability $\min\{1, \alpha x_i\}$
3. for each $i \in [m]$
   A. if $i$ is not covered by $F$
      1. add the cheapest set covering $i$ to $F$
4. return $F$

Show that for an appropriate choice of $\alpha$, 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.\(^4\)

\(^4\)Better yet, prove the same for $d \geq 2$. \n
114
Exercise 6.6. For CIPs, we could assume without loss of generality that \(A_{ij} \leq b_j\) for all \(i, j\). (In fact this assumption was critical for applying the Chernoff bound.) Suppose now that we had \(\lambda A_{ij} \leq b_j\) for all \(i, j\) for a parameter \(1 \leq \lambda \leq \log(n)\). Design and analyzing a \(O(\log(m)/\lambda)\)-approximation algorithm for this setting.

Exercise 6.7. In CIPs we allowed each \(x_j\) to be as large as we want. Suppose we added the constraint \(x_j \leq 1\) for all \(j\). Would the randomized rounding algorithm from section 6.3 still obtain a \((1 - 1/e)\)-approximation ratio? Why or why not?

Exercise 6.8. Recall the set cover problem for which we obtained a randomized \(O(\log n)\)-approximation. Here we consider a (maximum weight) set packing problem, defined as follows.

Let \([m]\) be a set of points, and let \(S_1, \ldots, S_n \subseteq [m]\) be \(n\) subsets of \([m]\). Let \(b_1, \ldots, b_n > 0\) represent the profit of \(S_1, \ldots, S_n\), respectively. We say that a collection of sets \(\mathcal{F} = \{S_{j_1}, \ldots, S_{j_k}\}\) is a set packing if they are all disjoint. The total profit of such a set packing is defined as the sum of profits \(b_{j_1} + \cdots + b_{j_k}\) of the corresponding sets.

The goal is to compute a set packing of maximum profit, but the problem is NP-Hard. Here we consider the following (perhaps unusual) approximation criteria. Let \(\text{OPT}\) denote the maximum profit of any set packing. For \(\alpha \geq 1\), we say that a collection of sets \(S_{j_1} + \cdots + S_{j_k}\) is an \(\alpha\)-packing if each point is covered by at most \(\alpha\) sets \(S_{j_h}\). We say that a randomized collection of sets \(\mathcal{F}\) is a randomized approximate \(\alpha\)-packing if

1. The expected total profit of \(\mathcal{F}\) is at least \(\text{OPT}\).
2. With high probability, \(\mathcal{F}\) is an \(\alpha\)-packing.

Design and analyze a polynomial time algorithm that outputs a randomized approximate \(\alpha\)-packing for \(\alpha\) as small as possible.\(^5\)

6.A Proof of the multiplicative Chernoff bound

Theorem 6.8. Let \(X_1, \ldots, X_n \in [0, 1]\) be independent random variables, and \(\epsilon \in [0, 1]\).

- For \(\mu \geq \mathbb{E}[X_1 + \cdots + X_n]\),
  \[
  \mathbb{P}[X_1 + \cdots + X_n \geq (1 + \epsilon)\mu] \leq e^{-\epsilon^2 \mu/3}.
  \]

\(^5\)Here we are interested in the approximation factor \(\alpha\) – the smaller and closer to 1 the better – and not the exact polynomial running time.
• For $\mu \leq \mathbb{E}[X_1 + \cdots + X_n]$, 

$$\mathbb{P}[X_1 + \cdots + X_n \leq (1 - \epsilon)\mu] \leq e^{-\epsilon^2 \mu/2}.$$ 

**Proof.** We prove the claim for a slightly weaker constant in the exponent. Let $t \in [0, 1]$ be a parameter TBD. We have 

$$\mathbb{P}[X_1 + \cdots + X_n \geq (1 + \epsilon)\mu] = \mathbb{P}[e^{t(X_1 + \cdots + X_n)} \geq e^{(1+\epsilon)\mu}] \leq \frac{\mathbb{E}[e^{t(X_1 + \cdots + X_n)}]}{e^{(1+\epsilon)\mu}}$$

by (a) Markov’s inequality. Next analyze $\mathbb{E}[e^{tX_1 + \cdots + X_n}]$. We first have 

$$\mathbb{E}[e^{tX_i}] = \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}]$$

by independence of the $X_i$’s. For each $i$, we have 

$$e^{tX_i} \leq 1 + tX_i + t^2 X_i^2 \leq 1 + (t + t^2)X_i$$

where we recall that $0 \leq t, X_i \leq 1$ and apply the inequality $e^x \leq 1 + x + x^2$ for $x \leq 1$. Consequently 

$$\mathbb{E}[e^{tX_i}] \leq \mathbb{E}[1 + (t + t^2)X_i] \leq 1 + (t + t^2) \mathbb{E}[X_i] \leq e^{(t+t^2)} \mathbb{E}[X_i],$$

where the last inequality follows from $1 + x \leq e^x$ for all $x$. Now we have 

$$\mathbb{E}[e^{t(X_1 + \cdots + X_n)}] \leq \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}] \leq \prod_{i=1}^{n} \mathbb{E}[e^{(t+t^2)} \mathbb{E}[X_i]] \leq e^{(t+t^2)\mu},$$

hence 

$$\mathbb{P}[X_1 + \cdots + X_n \geq (1 + \epsilon)\mu] \leq e^{(t+t^2)\mu - (1+\epsilon)\mu} = e^{(t-\epsilon)\mu}.$$ 

The RHS is minimized by setting $t = \epsilon/2$, giving $e^{-\epsilon^2 \mu/4}$.

The proof for the second inequality is similar and we provide a sketch highlighting a differences. Let $t \in (0, 1]$ by a parameter TBD. We have 

$$\mathbb{P}[X_1 + \cdots + X_n \leq (1 - \epsilon)\mu] = \mathbb{P}[e^{-t(X_1 + \cdots + X_n)} \geq e^{-(1-\epsilon)t\mu}]$$

$$\leq \mathbb{E}[e^{-t(X_1 + \cdots + X_n)}]e^{(1-\epsilon)t\mu}$$
Note the introduction of the negative sign to reverse the inequality before applying Markov’s inequality. As before, the apply independence to break the analysis into individual moments $E[e^{-tX_i}]$, which are each bounded above by

$$E[e^{-tX_i}] \leq 1 - \left(t - t^2/2\right) E[X_i] \leq e^{-(t-t^2/2)} E[X_i]$$

where we apply the inequality $e^x \leq 1 + x + x^2/2$ for $x \leq 0$. This leads to

$$P[X_1 + \cdots + X_n \leq (1 - \epsilon)\mu] \leq e^{(t^2/2 - \epsilon t)\mu};$$

The RHS is minimized by $t = \epsilon$. \hfill \Box
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\(^1\). We can store the set of different visitors in a dictionary, maybe even (or probably) 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 [HNH13], that explains some low-level enhancements to a well-known algorithm for the distinct elements problem called HyperLogLog [FFGM07].

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 [HNSS95]. 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 [AMS99] 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 functions \( h : \{n\} \rightarrow [0,1] \). 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. \( \mathbb{E}[X] = \frac{1}{k+1} \).
2. \( \mathbb{E}[X^2] = \frac{2}{(k+1)(k+2)} \).

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 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., [Bil12].)

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

$$P[a \leq X \leq b] = \int_a^b f(t) \, dt.$$ 

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.

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

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 $\alpha \geq 1$,

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

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

\(^3\)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 [Bil12].
7. Distinct elements

7.3. Distinct elements, continued

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

One can use the idealized, continuous hash function \( h : [m] \to [0, 1] \) 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

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

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 than $\epsilon/(k + 1)$ 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

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

Consider the numerator. We have


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.) Chebyhev’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 \( \overline{X} = \frac{1}{\ell} \sum_{i=1}^\ell X_i \) be the average of the \( X_i \)'s. Let \( V = \frac{1}{\ell} \sum_{i=1}^\ell \text{Var}[X_i] \) be the average variance over the \( X_i \)'s. Then

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

Proof. We have

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

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

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

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

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

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

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

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

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

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

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 \( h_1, \ldots, h_\ell : [m] \to [0, 1] \). 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 of 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.

\footnote{The 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 median trick, or the median of means. By taking the median of means, we can efficiently reduce the error probability at an 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 fuction. We return $r/X$ as our estimate for the number of distinct elements.

For additional details, see [Nel20, §2.2.2] or [Che14a, §3.2].

### 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 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 then the mean.

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

distribution $X$ w/ mean $\mu$

variance $\sigma^2$

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

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

$(1-\epsilon)\mu \leq s_t \leq (1+\epsilon)\mu$

$w/\text{prob } 1-\delta.$

$w=O\left(\frac{1}{\epsilon^2} \cdot \frac{\sigma^2}{\mu^2}\right)$

\[ \begin{aligned}
& X_{i,1}, X_{i,2}, \ldots, X_{i,w} \xrightarrow{\text{mean}} Y_{i,1} \\
& X_{2,1}, X_{2,2}, \ldots, X_{2,w} \xrightarrow{\text{mean}} Y_{i,2} \\
& \vdots \\
& X_{d,1}, X_{d,2}, \ldots, X_{d,w} \xrightarrow{\text{mean}} Y_{i,d} \\
\end{aligned} \]

\[ \xrightarrow{\text{median}} Z \]

Figure 7.1: The median trick.
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 \( r \approx 1/\epsilon^2 \), 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.

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

(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 $\pm \epsilon \|x\|_2$ with high probability, in sublinear space.

Below we describe a data structure that can get $\pm \epsilon \|x\|_2$ error for an appropriate choice of parameters. (This is like describing one “row” of the count-min data structure.) You will first be asked to choose the parameters and prove the error guarantee. Then you will be asked to amplify the data structure to obtain a high probability guarantee.

The data structure is as follows. Let $\epsilon > 0$ be given, let $w \in \mathbb{N}$ be a parameter TBD, and let $A[1..w]$ be an array of values initially set to 0. Let $h : [n] \to [w]$ be a pairwise independent hash function. Let $g : [n] \to \{-1, 1\}$ be a second pairwise independent hash function. The operations are as follows.

- For each update $(i, \Delta)$ 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. What is the variance of $y_i$ (as a function of $w$)?

3. Prove that for an appropriate choice of $w$, the probability that $|x_i - y_i| \geq \epsilon \|x\|_2$ is at most 1/10. ($w$ should depend on $\epsilon$, 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.)

4. Using the data structure designed above, design and analyze a data structure that, in $O(\log(n)/\epsilon^2)$ space, estimates each coordinate of $x$ up to an additive error of $\pm \epsilon \|x\|_2$ with high probability. (I.e., probability of error at most $1/\text{poly}(n)$.)
**Exercise 7.6.** You run a double-secret laboratory and for your experiments you need to monitor the temperature of the lab very carefully. To this end you can buy thermostats $T_1, \ldots, T_k$ that purport to measure the temperature $\mu$, but the thermostats are imperfect. You have the following facts.

1. Given the actual temperature $\mu$ of the lab, the thermostat readings $T_i$ are independent.

2. Each thermostat is calibrated so that its expected value $\mathbb{E}[T_i]$ equals the actual temperature $\mu$ of the lab.

3. For each thermostat $T_i$, the variance $\text{Var}[T_i]$ of the thermostat is $\sigma^2$ for a known parameter $\sigma > 0$.

Given parameters $\epsilon, \delta \in (0, 1)$, the goal is to be able to measure the temperature of the room with additive error at most $\epsilon$ with probability at least $1 - \delta$. Describe and analyze a system that, using as few thermostats as possible\footnote{up to constant factors independent of $\epsilon, \delta, \text{ and } \sigma$}, obtains additive error $\epsilon$ with probability at least $1 - \delta$. 
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 [MSCCD13].

We note that in some of the examples above, the data vectors are typically sparse with few nonzero entries. Such vectors can be represented more compactly as an “adjacency list”, where we list the index and the value of only the nonzero entries in the vector. The trouble arises when we start running computations over them. When we start combining these vectors in some linear algebraic procedure, the vectors
rapidly become dense, and this is where we pay for the high dimensions.

Most operations with vectors take time proportional to the number of dimensions (in the worst and dense case). Certainly it would be desirable for the data to live in a much lower dimensional space. The goal in this discussion is to develop some techniques for transforming high-dimensional data into lower-dimensional data. We first note that for many applications, we do not necessarily require the exact coordinates of the vector. Given a set \( P \) of points in a high-dimensional space \( d \), we may only actually need the following:

1. For a given point \( x \in P \), the (Euclidean) length of \( x \), \( \|x\| = \sqrt{\sum_i x_i^2} \).
2. For any two points \( x, y \in P \), the Euclidean distance \( \|x - y\| \) between them.
3. For any two points \( x, y \in P \), the dot product \( \langle x, y \rangle \) between \( x \) and \( y \).

Moreover, for many applications, approximations to the above quantities may suffice to produce approximation algorithms for the given context.

We now introduce the main result of this chapter.

**Theorem 8.1** (Johnson and Lindenstrauss [JL84]). Let \( P \subseteq \mathbb{R}^d \) be a collection of \( n \) points in \( \mathbb{R}^d \), and let \( k = O(\log(n)/\epsilon^2) \). Let \( A \in \mathbb{N}^{k \times d} \) be a \( k \times d \) randomized matrix where each coordinate is sampled as an independent Gaussian. Consider the randomly constructed linear map \( f : \mathbb{R}^d \to \mathbb{R}^k \) defined by

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

Then with probability of error \( \leq 1/poly(n) \), 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 formiddably expensive. Here one can instead replace the Gaussian entries with $\{-1, 0, 1\}$ entries generated by appropriate hash functions [Ach01; CJN18; DKS10; KN14]. The intuition is similar because $\{-1, 0, 1\}$-random variables behave similarly to Gaussian’s in a certain technical sense. (They are both sub-Gaussian; see [Ver18]). There is particular interest in ensuring that $A$ is column-sparse, since this determines the running time of applying $A$. An alternative approach uses Hadamard matrices to produce a version of $A$ that can be applied extremely quickly [AC09].

An important application of dimensionality reduction is in accelerating numerical algorithms on large matrices. See for example [DM17; Woo14a].

### 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 $N(\mu, \sigma^2)$ is parametrized by two parameters $\mu$ and $\sigma^2$. We write $x \sim N(\mu, \sigma^2)$ 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 $x \sim N(\mu, \sigma^2)$. Then the mean and variance of $x$ are

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

We abbreviate $N \overset{\text{def}}{=} N(0, 1)$ 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

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

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

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 \( \mu = \mathbb{E}[\|x\|^2] = k \).

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

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

(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

\[
\alpha = \frac{1}{1+2t} \iff t = \frac{1 - \alpha}{2\alpha}.
\]  \( (8.1) \)

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

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

as desired.

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

\[
P[\|x\|^2 \geq \alpha k] = P[e^{t\|x\|^2} \geq e^{t\alpha k}] \leq \mathbb{E}[e^{t\|x\|^2}] e^{-t\alpha k} \leq \left( \frac{1}{1-2t} \right)^{k/2} e^{-t\alpha k} = \exp\left( -\frac{k}{2} (2\alpha t + \ln(1-2t)) \right)
\]

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

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

as desired. \( \square \)
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) $P\left[\|x\|^2 \leq (1 - \epsilon) E[\|x\|^2]\right] \leq e^{-c\epsilon^2 k/2}$ for $c = 1 - \ln(2) \approx 0.307$.

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

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

$P\left[\|x\|^2 \geq (1 + \epsilon) k\right] \leq (1 + \epsilon)e^{-\epsilon} < e^{c\epsilon^2 k/2}$.

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

Likewise, we have

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

(c) is by fact 8.6. (d) is because $1 - \epsilon \leq e^{-\epsilon - c\epsilon^2/2}$ 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 $x$,

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

**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.
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
\[
(1 - \epsilon)\|x\|^2 \leq \|f(x)\|^2 \leq (1 + \epsilon)\|x\|^2 \tag{8.2}
\]
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
\[
f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x - \mu}{\sigma})^2}. \tag{8.3}
\]
(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

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

Then we have the following.

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

$$\left( \int_{-\infty}^{\infty} e^{-x^2/2} \, dx \right)^2 = \int_{-\infty}^{\infty} e^{-(x^2+y^2)/2} \, dx \, dy = \int_0^{2\pi} \int_0^\infty r e^{-r^2/2} \, dr \, d\theta = 2\pi \int_0^\infty r e^{-r^2/2} \, dr = 2\pi.$$

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

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

For the third claim, we have

$$\int_{-\infty}^{\infty} x^2 e^{-x^2/2} \, dx = \left[ -xe^{-x^2/2} \right]_{-\infty}^{+\infty} + \int_{-\infty}^{\infty} e^{-x^2/2} = 1$$

by (a) integration by parts. \qed

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

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

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

**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

$$f(Ux) \equiv (2\pi)^{-n/2}e^{-\langle Ux,Ux \rangle/2} \equiv (2\pi)^{-n/2}e^{-\langle x,x \rangle/2} \equiv f(x),$$

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

**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 $u = U^Te_1$ for an orthogonal matrix $U$. Then

$$\langle u, x \rangle = \langle U^Te_1, x \rangle = \langle e_1, Ux \rangle = (Ux)_1 \stackrel{(a)}{\sim} \mathcal{N},$$

where (a) is by lemma 8.11. □

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 $x_1 + x_2 \sim \mathcal{N}(\mu_1 + \mu_2,\sigma_1^2 + \sigma_2^2)$. 

141
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

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

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 $|x| \leq c_1$,

$$
1 + x \leq e^{x-c_2x^2}.
$$

(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 $f : \mathbb{R}^d \to \mathbb{R}^k$ 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

\[|\langle f(x), f(y) \rangle - \langle x, y \rangle| \leq \epsilon\|x\|\|y\|\]

for all $x, y \in P$.\(^1\)

Exercise 8.4. Let $P \in \mathbb{R}^{\ell \times n}$ be the random projection function as described in theorem 8.1, for a parameter $\ell$ to be determined. We want to argue that for $\ell = O(k/\epsilon^2)$, $P$ approximately preserves all of the vectors of a fixed subspace $U$ of dimension $k$ with high probability (in $k$).

To express this more formally, let $U$ be a fixed (but unknown) subspace of $\mathbb{R}^n$ of dimension $k$. We claim that, with probability at least $1 - e^{-O(k)}$, we have

\[(1 - \epsilon)\|x\|^2 \leq \|Px\|^2 \leq (1 + \epsilon)\|x\|^2\] for all $x \in U$ (simultaneously). \(^{(8.4)}\)

Note that the algorithm does not know $U$; for this reason, $P$ is called an $(1 \pm \epsilon)$-approximate oblivious subspace embedding. Oblivious subspace embeddings are useful for developing faster approximation algorithms in numerical linear algebra.

In this exercise we prove that $P$ is an oblivious subspace embeddings with high probability. Let $U$ be a subspace of dimension $k$. Let $\mathbb{S}^k$ be the unit sphere in the $k$-dimensional subspace we want to preserve. Since the requirements of (8.4) are scale invariant, it suffices to establish (8.4) for all points in $\mathbb{S}^k$.

Our argument makes use of a geometric technique called $\epsilon$-nets. For a set $S$, an $\epsilon$-net is a set $N$ such that for every point $s \in S$, there is a point $x \in N$ such that $\|x - s\| \leq \epsilon$.\(^2\) We need to show that there exists a relatively small $\epsilon$-net for $\mathbb{S}^k$.

\(^1\)It might be helpful to work through the special case where $\|x\| = \|y\| = 1$.
\(^2\)This is similar but different from the $\epsilon$-nets in chapter 12.
1. Let $N \subseteq S^k$ be a maximal set of points such that any two points in $N$ have distance at least $\epsilon$. Show that $N$ is an $\epsilon$-net, and that $N$ has at most $(1 + 2/\epsilon)^k$ points.\(^3\)

Let $N$ be an $(1/2)$-net of $S^k$ with at most $5^k$ points. Next we establish that we preserve the length of all points in $N$ is preserved with high probability.

2. Show that with probability $1 - e^{-O(k)}$, we have $\|Px\|^2 - 1 \leq 1/2$ for all $x \in N$, and $|\langle Px, Py \rangle - \langle x, y \rangle| \leq 1/2$ for all $x, y \in N$.

So now we know that we preserve all points of $N$ with high probability. We want to argue that this suffices to preserve all the vectors.

3. Prove that for any unit vector $x \in S^k$, one can write $x = x_0 + x_1 + x_2 + \cdots$ such that for all $i$:
   
   (a) $\|x_i\| \leq 2^{-i}$.
   
   (b) $x_i/\|x_i\| \in N$.\(^4\)

Now use the representation above to prove that $P$ is an oblivious subspace embedding.

4. Prove that (8.4) holds with probability at least $1 - e^{-O(k)}$.

The high-level takeaway from the proof is that if you can embed an $\epsilon$-net of the unit sphere for constant $\epsilon$, then you automatically embed the entire subspace.

\(^3\)Hint: $N$ packs $|N|$ interior-disjoint $k$-dimensional balls of radius $\epsilon/2$ into an $k$-dimensional ball of radius $1 + \epsilon/2$. It is helpful to know that the volume of an $n$-dimensional ball of radius $r$ is $c_n r^n$ for a parameter $c_n > 0$ depending on $n$.

\(^4\)Hint: Choose $x_0$ to be the closest point in $N$ to $x$. Observe that $\|x - x_0\| \leq 1/2$ because $N$ is a $(1/2)$-net. It remains to express $x - x_0$ as the sum $x_1 + x_2 + \cdots$. How might you choose $x_1$?
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 high-dimensional vectors. Surprisingly, 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 [IM98].\(^1\) 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

\(^1\)Here the author would like to propose the term *clashing*. 

146
all the points in our data set $P$, as well as any query point, is normalized to lie on $S^{d-1}$. We describe an LSH scheme due to [Cha02].

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 $h : S^{d-1} \to \{-1, 1\}^k$ where each coordinate is generated by splitting $S^{d-1}$ 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 $h : S^{d-1} \to \{-1, 1\}^k$ 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 $x, y \in S^{d-1}$, and let $g \sim \mathcal{N}^d$. Then

$$\mathbb{P}[\text{sign}(\langle g, x \rangle) \neq \text{sign}(\langle g, y \rangle)] = \frac{\angle(x, y)}{\pi}.$$ 

**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

$$\mathbb{P}[h(x) = h(y)] = \left(1 - \frac{\angle(x, y)}{\pi}\right)^k \approx e^{-\frac{\angle(x, y)}{\pi}k} = e^{-\log(n)/2} = \frac{1}{\sqrt{n}}.$$ 

On the other hand, if $\angle(x, y) > 2\theta$, then

$$\mathbb{P}[h(x) = h(y)] = \left(1 - \frac{\angle(x, y)}{\pi}\right)^k \leq e^{-\angle(x, y)k/\pi} \leq e^{-\log(n)} = \frac{1}{n}.$$ 

Thus, when querying a point $y$: 

147
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 \( O(dn^{3/2} \text{polylog}(n)) \) 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 \( h : \mathbb{R}^d \to \mathbb{Z}^k \) 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 \( h : \mathbb{R}^d \to \mathbb{Z} \) 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 \( h : \mathbb{R}^d \to \mathbb{Z} \) by the function

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

\(^2\text{a clash function?}\)
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 \( x, y \in \mathbb{R}^d \).

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

**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

\[
\mathbb{P}[h(x) = h(y)] = 2 \int_0^1 \mathbb{P}[h(x) = h(y) | |\langle g, x - y \rangle| = t]f(\|x - y\|t) dt
\]

\[
= 2 \int_0^1 (1 - t)f(\|x - y\|t) dt,
\]

as desired.

**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

\[
p = 2 \int_0^1 (1 - t)f(t) dt
\]
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 by amplification, as follows.

9.3.2 Amplifying the gap

Let $k \in \mathbb{N}$ be a parameter TBD. We define a hash function

$$h : \mathbb{R}^d \to \mathbb{R}^k$$

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, increasing $k$ decreases both $p^k$ and $q^k$, which is both good and bad. As $p^k$ decreases, so do the odds of finding a near neighbor when we hash. We will need to rebuild the data structure $\ell = 1/p^k$ times to be able to find a good neighbor with
constant probability, which is expensive.\textsuperscript{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 $(q/p)^k n$ 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

$$\left( \frac{1}{p} \right)^k (k q^k n).$$

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 than 1. In fact, one can show that it is about $1/1 + \epsilon$ for $\sigma \approx 1 + \epsilon$.

**Theorem 9.6.** One can compute a $\rho$-approximate nearest neighbor w.r.t Euclidean distance with high probability in $\tilde{O}(dn^{\rho(\sigma)})$ 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

$$\rho(\sigma) \leq 1/\sigma^2.$$

\textsuperscript{3}Indeed, the probability of failing to find the good neighbor in each of $\ell$ constructions, each of which has $k$ hash coordinates, is

$$(1 - p^k)^\ell \approx e^{-p^k\ell}.$$
9.3.3 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 [DIIM04]. 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

$$\sqrt{2\pi}(p - q) = \int_0^1 (1 - t)\left(e^{-t^2/2} - e^{-\sigma^2t^2/2}\right)$$

Say $\sigma^2 = (1 + \epsilon)$ with $\epsilon$ sufficiently small. Then for all $t \in [0, 1]$, we have

$$e^{-t^2/2} - e^{-(1-\epsilon)t^2/2} = e^{-t^2/2}\left(1 - e^{-\epsilon t^2/2}\right)$$

$$= e^{-t^2/2}\left(\epsilon t^2/2 - \left(\epsilon t^2/2\right)^2/2\right) \leq \epsilon(1 - \epsilon)e^{-t^2/2}.$$  

Thus

$$\int_0^1 (1 - t)\left(e^{-t^2/2} - e^{-t^2/2\sigma^2}\right) dt \leq \epsilon(1 - \epsilon) \int_0^1 (1 - t)e^{-t^2/2} dt$$

$$= \epsilon(1 - \epsilon)\sqrt{2\pi}p.$$  

Thus

$$\sqrt{2\pi}(p - q) \leq \epsilon(1 - \epsilon)\sqrt{2\pi}p;$$

in turn;

$$(1 - \epsilon)^2 p \leq q.$$  

\qed

152
Lemma 9.8. Let $\sigma^2 \leq 1.6$. Then
\[
\rho(\sigma) \leq \frac{1}{\sigma^2}.
\]

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 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.

9.5 Exercises

Exercise 9.1. In this exercise, we will develop a 2-approximate LSH scheme for bit strings $s \in \{0, 1\}^d$ of a fixed length $d$ with respect to Hamming distance. The Hamming distance between two strings $s, t \in \{0, 1\}^d$ is 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 $h : \{0, 1\}^d \rightarrow \{0, 1\}$ 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 $\mathbb{P}[h(s) = h(t)]$, as a function of the Hamming distance between $s$ and $t$?
2. Fix a target distance $r \in [0, 1]$. Construct a data structure that over a set $P$ of $n$ strings $\{0, 1\}^d$ to answer the following query with high probability.

*Given a query point $s \in \{0, 1\}^d$, either return a point $x \in P$ with Hamming distance $\leq 2r$ from $s$, or declare that there are no points within Hamming distance $r$ from $s$.*

In addition to describing the algorithm, one should analyze the preprocessing time and space, the query time, and the probability of correctness.

3. Briefly describe how to use the above data structure to efficiently find 2-approximate nearest neighbors with respect to Hamming distance (with high probability).
Chapter 10

$L_1$-metric embeddings and sparsest cut

10.1 LP duality, line embeddings, and max-flow min-cut

The reader may (should) have seen max flow, the max-flow min-cut theorem, and some algorithms for max flow in an introductory class on algorithms. Here we present these results from a different perspective that primes us for the techniques used to approximate the sparsest cut problem.

10.1.1 Packing and covering paths

Let $G = (V, E)$ be a directed graph\(^1\), and let $s, t \in V$ be two distinct vertices. We call $s$ the source and $t$ the sink. A path packing is a collection of edge disjoint paths. An $(s, t)$-path packing is a path packing of $(s, t)$-paths. The maximum $(s, t)$-path packing problem\(^2\) is to

find a maximum cardinality packing of $(s, t)$-paths.

An $(s, t)$-cut is a set of edges whose removal disconnects $s$ from $t$. The minimum $(s, t)$-cut problem is to

---

1. $G$ is allowed to be a multi-graph, with multiple copies of the same edge.

2. Better known as the uncapacitated maximum $(s, t)$-flow problem for reasons we discuss later.
find the minimum cardinality \((s, t)\)-cut.

Both can be understood as a generalization of reachability. Reachability is concerned with whether there is a single connection from \(s\) to \(t\). Both the maximum flow and minimum cut problems measure the strength of the connection from \(s\) to \(t\).

We are interested in these questions both algorithmically and (graph-)structurally. Algorithmically the problems are highly non-trivial, as there are exponentially many possible paths from \(s\) to \(t\) to take into account. It is not obvious that there is a polynomial time algorithm for either problem.

10.1.2 Duality.

The \((s, t)\)-path packing and \((s, t)\)-cut problem are dual packing and covering problems, in the following sense. We are selecting paths that “pack in” to the edges of the graph – each path uses up all of its edges. Conversely, an \((s, t)\)-cut must contain at least one edge from every \((s, t)\)-path. That is, we are trying to “cover” the paths with edges, where we interpret each edge as a set that covers all the \((s, t)\)-paths that contain that edge. In short:

paths pack into edges, and edges cover paths.

As with any packing and covering problem, we have the following inequality.

\[
\text{(max \((s, t)\)-path packing)} \leq \text{(min \((s, t)\)-cut).}
\]

Indeed, let \(P\) denote a path packing, and let \(C \subseteq E\) be an \((s, t)\)-cut. We have

\[
|P| = \sum_{p \in P} 1 \leq \sum_{p \in P} |p \cap C| \leq |C|.
\]

Above, we treat each path \(p \in P\) as a subset of edges. (a) is because, as an \((s, t)\)-cut, \(C\) contains at least one edge from every \((s, t)\)-path. (b) is because the paths \(p \in P\) are edge disjoint, so the sets \(p \cap C\) over \(p \in P\) are also disjoint. The following is a conceptual sketch of our argument.

\[
\begin{align*}
\text{paths pack into edges, and edges cover paths.}
\end{align*}
\]

The inequality (10.1) inspires some basic questions. Is the inequality ever equal? Is the inequality ever strict?

We now introduce some variations of the \((s, t)\)-paths and cut problems.
10.1.3 Capacities and costs.

A natural generalization of disjoint \((s, t)\)-paths allows edges to be reused, and extends the input to include edge capacities \(c : E \to \mathbb{R}_{>0}\) that set a numerical limit on how many times each edge can be used. For example, if an edge \(e\) has \(c(e) = 2\), then this implies we are allowed to use \(e\) twice (i.e., as if there are two copies of \(e\)). The same path is allowed to be selected multiple times. Formally, the problem becomes:

Find a collection of paths \(P\) of maximum cardinality \(|P|\) such that each edge \(e\) is contained in at most \(c(e)\) paths.

We denote the problem formalized above as \((\text{Max-Paths})\). Abusing notation, we will also let \((\text{Max-Paths})\) denote the optimum objective value of the \((\text{Max-Paths})\) problem.

Introducing edge capacities makes it even more challenging to find a polynomial time algorithm. Before, without edge capacities, it is clear that the optimum solution has a polynomial number of paths since each path must use up at least one edge. Once we introduce capacities – which may be large numbers expressed in a logarithmic number of bits – we can no longer assume that the maximum path packing has polynomial size!

Flip back to the dual \((s, t)\)-cut problem. A natural generalization introduces positive edge costs \(c : E \to \mathbb{R}_{>0}\); the problem becomes:

Find an \((s, t)\)-cut \(C \subseteq E\) of minimum cost \(\sum_{e \in C} c(e)\). \((\text{Min-Cut})\)

The minimum cardinality cut problem is equivalent to the minimum cost cut problem with uniform costs \((c(e) = 1\) for all \(e\)). In contrast to edge capacities, edge costs do not invoke the risk of the optimum solution no longer being compact.

As with \((\text{Max-Flow})\), we write \((\text{Min-Cut})\) to denote both the problem described above and the optimum value of that problem.

When the capacities and costs are based on the same vector \(c\), then \((\text{Max-Paths})\) and \((\text{Min-Cut})\) are dual\(^3\) to one another. In particular one can show that \((\text{Max-Paths}) \leq (\text{Min-Cut})\) by a similar argument as before in the uncapacitated/uniform-cost setting. We leave the proof to the reader.

**Lemma 10.1.** Let \(G = (V, E)\) be a directed graph and \(s, t \in V\). Let \(c : E \to \mathbb{R}_{>0}\) be a fixed set of capacities / costs. Then

\((\text{Max-Paths}) \leq (\text{Min-Cut})\).

\(^3\)We will formalize the definition of dual soon.
10.1.4 Fractionally packing and covering paths.

Another variation of the above problems allow for fractional solutions. Consider first path packings. Let $\mathcal{P}_{s,t}$ denote the family of all $(s,t)$-paths. A fractional collection of paths is an assignment $x : \mathcal{P}_{s,t} \to \mathbb{R}_{\geq 0}$ giving nonnegative weight to every path. A fractional path packing is a fractional collection of paths $x$ that satisfies the capacity constraint $c(e)$ for each edge $e$ in the following quantitative sense:

$$\sum_{p \ni e} x_p \leq c(e).$$

(Here “$p \ni e$” is summing over all $p \in \mathcal{P}_{s,t}$ such that $e \in p$.) Subject to this constraint, the goal is to find the fractional path packing of maximum total quantity,

$$\sum_{p \in \mathcal{P}_{s,t}} x_p.$$

Putting it all together, the entire fractional path packing problem is given by:

$$\begin{align*}
\text{maximize} \quad & \sum_{p \in \mathcal{P}_{s,t}} x_p \\
\text{over} \quad & x : \mathcal{P}_{s,t} \to \mathbb{R}_{\geq 0} \\
\text{s.t.} \quad & \sum_{p \ni e} x_p \leq c(e) \text{ for all } e \in E.
\end{align*}$$

(Max-Flow)

Note that the objective ($\sum_p x_p$) is a linear function of $x$, and the constraints ($x_p \geq 0$, $\sum_{p \ni e} x_p \leq c(e)$...) are linear inequalities. This feature is very important and we will return to it in a moment.

Due to the continuous nature of the fractional path packing problems – imagine $x_p$ units of water flowing along the path $p$ – it is commonly referred to as the maximum flow problem. For the rest of this section, we let (Max-Flow) refer to the maximum flow problem formulated above. Abusing notation, we will also let (Max-Flow) refer to the optimum value of (the problem) (Max-Flow). We also note that the (discrete) path packing is also called integral maximum flow.

Compare max flow with the (integer) path packing problem discussed above, where we can only select integer multiples of paths. Clearly, any integer solution is a feasible solution to the fractional version. For this reason, the fractional path packing problem is called a relaxation of the integer path packing problem – every feasible solution to the latter is feasible in the former. As a relaxation of a maximization problem, we always have (Max-Flow) $\geq$ (Max-Paths).

We can apply the same fractional perspective to the minimum cost $(s,t)$-cut problem. Recall that an $(s,t)$-cut contains at least one edge from every $(s,t)$-path. A
fractional \((s,t)-\text{cut}\) is a fractional combination of edges \(y : E \rightarrow \mathbb{R}_{\geq 0}\) that contains (in sum) one unit of edges from every \((s,t)\)-paths. An edge cost \(c(e)\) are now interpreted as the cost of one unit \(y(e)\). All put together, the fractional relaxation of minimum cost \((s,t)\)-cut is given by the following problem:

\[
\text{minimize } \sum_{e \in E} c(e)y_e \text{ over } y : E \rightarrow \mathbb{R}_{\geq 0} \text{ s.t. } \sum_{e \in p} y_e \geq 1 \text{ for all } p \in \mathcal{P}_{st}.
\]  

(10.2) has \(m\) variables but exponentially many constraints. This setup leads to the following situation. We encourage the reader to pause and consider the following question herself before reading on.

Suppose you were given a vector \(y \in \mathbb{R}^E_{\geq 0}\). How would you verify, in polynomial time, that \(y\) is a feasible solution? In particular, how does one verify that for every \((s,t)\)-path \(p\), the sum of \(y_e\)'s over \(e \in p\) is at least 1? (Is it even possible?)

The question is nontrivial because there is not enough time to enumerate every \((s,t)\)-path. But let us reformulate the question slightly: verifying every path \(p\) has \(\sum_{e \in p} y_e \geq 1\) is the same as verifying that the minimum \(\sum_{e \in p} y_e\), over all \(p \in \mathcal{P}_{st}\), is at least 1. Let us reinterpret the values \(y : E \rightarrow \mathbb{R}_{\geq 0}\) as edge lengths. Then the covering constraint is really saying that the length of the shortest \((s,t)\)-path w/r/t edge lengths \(y_e\), is \(\geq 1\); we can verify this constraint by computing the shortest \((s,t)\)-path w/r/t \(y\).

An equivalent formulation of (10.2), then, is as follows.

Find the minimum cost set of edge lengths \(y : E \rightarrow \mathbb{R}_{\geq 0}\) subject to \(s\) and \(t\) having distance 1 in the shortest path metric induced by \(y\).

This problem, besides being a relaxation of \((s,t)\)-cut, is a very natural problem in its own right. For this reason, and to help distinguish the continuous nature of (10.2) for the discrete min-cut problem, we will also refer to the fractional min-cut problem as the \((s,t)\)-minimum cost metric problem. We will write (Min-Metric) to denote both the optimization problem and the value of the optimization problem formulated in (10.2) above.

10.1.5 Linear programming and LP duality

The fractional versions of the \((s,t)\)-path packing and cut problem described above are examples of \textit{linear programs}, a class of mathematical optimization problems previously introduced in chapter 6. We briefly review the basics.
Linear programs (LP’s) are *constrained* continuous optimization problems where the goal is to (a) select a vector \( x \in \mathbb{R}^n \) that (b) optimizes a linear objective subject to (c) linear equality and inequality constraints. That is, an optimization problem of the form

\[
\begin{align*}
\min / \max \ & \langle b, x \rangle = \sum_{j=1}^{n} b_j x_j \text{ over } x \in \mathbb{R}^n \\
\text{s.t.} \ & A_1 x \leq c_1, \ A_2 x = c_2, \text{ and } A_3 x \geq c_3.
\end{align*}
\]

where \( A_1, A_2, A_3 \) are matrices and \( b, c_1, c_2, c_3 \) are vectors.

Clearly, linear programs are useful for modeling real problems where we seek continuous solutions. Throughout this class we will encounter many different uses for LP’s for understanding and solving *discrete* problems as well. A powerful feature of LP’s is that they are *polynomial time solvable* \(^4\), and conceptually it is easy to interact with these solvers as a black box. Moreover, real-world software for LP’s is well-developed and reliable in practice.

We now introduce two canonical classes of LP’s that capture most combinatorial problems.

**Packing LPs.** A *packing LP* is a linear program of the form

\[
\max \ \langle b, x \rangle \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } Ax \leq c. \quad (P)
\]

where \( A \in \mathbb{R}^{m \times n}_{\geq 0} \), \( b \in \mathbb{R}^n_{> 0} \), and \( c \in \mathbb{R}^m_{> 0} \) all have nonnegative coefficients. We let \( \text{OPT}(P) \) denote the optimum value of the LP \((P)\).

The fractional path packing problem is our first example of an packing LP. For path packing, we have one variable for every path. Identifying edges and paths as coordinates, then, we have:

1. \( b = 1^{\mathcal{P}_{s,t}} \), the all-ones vector in \( \mathbb{R}^{\mathcal{P}_{s,t}} \)
2. \( c \in \mathbb{R}^E_{> 0} \) is the edge capacities.
3. \( A \in \{0, 1\}^{E \times \mathcal{P}_{s,t}} \) is the incidence matrix defined by

\[
A_{e,p} = \begin{cases} 
1 & \text{if } e \in p \\
0 & \text{if } e \notin p,
\end{cases}
\]

for each edge \( e \in E \) and path \( p \in \mathcal{P}_{s,t} \).

\(^4\)More precisely, they are weakly polynomial time solvable, meaning the running times are polynomial in the bit complexity of the input.
Note that there are exponentially many variables in this LP so we could not even write it down in full in polynomial time, let alone apply a black box LP solver. Fortunately there are otherwise to solve the LP, as we will see.

**Covering LPs.** A *covering LP* is a linear program of the form

\[
\min \langle c, y \rangle \text{ over } y \in \mathbb{R}_+^m \text{ s.t. } A^T y \geq b,
\]

where \( A \in \mathbb{R}^{m \times n}_+ \), \( b \in \mathbb{R}^n_+ \), and \( c \in \mathbb{R}^m_+ \). We let \( \text{Opt}(C) \) denote the optimum value of the LP \((C)\).

The minimum cost metric problem above is our first example of a covering LP. For minimum cost metric, we have one variable/column for each edge, and one row/constraint for each \((s, t)\)-path.

1. \( c \in \mathbb{R}^E_+ \) is the edge costs.
2. \( b = 1_{\mathcal{P}_{st}} \) is the all-ones vector in \( \mathbb{R}^{\mathcal{P}_{st}} \).
3. \( A^T \in \{0, 1\}^{\mathcal{P}_{st} \times E} \) is the \( \{0, 1\} \)-incidence matrix defined by

\[
A^T_{p,e} = A_{e,p} = \begin{cases} 
1 & \text{if } e \in p \\
0 & \text{if } e \notin p,
\end{cases}
\]

for each edge \( e \in E \) and path \( p \in \mathcal{P}_{st} \).

Note that \( A, b, c \) are the same between our two examples.

**LP duality.** LP duality is about the relationship between the linear programs \((P)\) and \((C)\), particular when the matrices and vectors \( A, b, c \) are the same for both problems. In this case \((P)\) and \((C)\) are said to be dual to one another.

Suppose we have dual pair of \((P)\) and \((C)\); i.e., \( A, b, c \) refer to the same objects in either problem. Let \( x \in \mathbb{R}^n_+ \) be any feasible solution to \((P)\) and let \( y \in \mathbb{R}^n_+ \) be any feasible solution to \((C)\). We have

\[
\langle b, x \rangle \leq \langle A^T y, x \rangle \leq \langle y, Ax \rangle \leq \langle y, c \rangle.
\]

Here (a) is because \( x \geq 0 \) and \( A^T y \geq b \). (b) is by definition of the transpose. (c) is because \( y \geq 0 \) and \( Ax \leq c \). Thus, for a packing problem \((P)\) and a covering problem \((C)\) linked by duality, we have

\[
\text{Opt}(P) \leq \text{Opt}(C).
\]

---

\(^5\)Of course, in \((C)\), we could have written \( A \) instead of its transpose \( A^T \), and swapped \( b \) and \( c \), which would more closely resemble \((P)\). It is convenient for the subsequent discussion on LP duality for \( A, b \) and \( c \) to have the same dimensions in \((P)\) and \((C)\).
If this argument seems familiar, it is because we just saw it for packing and covering paths in section 10.1.2 above.

We ask the same question for packing and covering LP’s as we did for packing and covering paths. When, if ever, is $\text{Opt}(P) = \text{Opt}(C)$? The all-important LP duality theorem (here restricted to packing and covering problems) states that in fact they are always equal.

**Theorem 10.2** (LP Duality for packing and covering). $\text{Opt}(P) = \text{Opt}(C)$.

We note that theorem 10.2 holds even if $A$, $b$, and $c$ have negative coefficients.

We will see that LP duality has important consequences for many combinatorial problems of interest – starting with max flow in the present discussion. Recall that (Max-Flow) is a packing LP and (Min-Metric) is a covering LP. Moreover, they are dual to one another. The LP duality theorem then tells us that (Max-Flow) = (Min-Metric). As a relaxation, we also have that (Min-Metric) $\leq$ (Min-Cut). That is:

$$(\text{Max-Flow}) = (\text{Min-Metric}) \leq (\text{Min-Cut}).$$

### 10.1.6 Max-flow min-cut via LP duality

We now prove the following well-known max-flow min-cut theorem.

**Theorem 10.3** (Ford and Fulkerson [FF56] and Menger [Men27]). (Max-Flow) = (Min-Cut).

Typically this theorem is proven algorithmically by the Ford-Fulkerson algorithm. Here we given an alternative proof based on LP duality\(^6\). Having already established that (Max-Flow) = (Min-Metric) $\leq$ (Min-Cut)\(^7\), it suffices to prove that (Min-Cut) $\leq$ (Min-Metric).

\[^6\text{For a video, see https://youtu.be/J4yUdABv1tE.}\]
\[^7\text{With very little effort, thanks to LP duality.}\]
Given a fractional min-cut \( y \), we can find a discrete \((s, t)\)-cut \( C \subseteq E \) with total capacity, \( \sum_{e \in C} c(e) \), less than or equal to the fractional capacity of \( y \), \( \langle c, y \rangle \).

This is our first example of rounding a fractional solution to a discrete one.

**Notation.** Before proceeding, we introduce some standard notation for cuts. For a set of vertices \( S \subseteq V \), the directed out-cut of \( S \), consisting of edges leaving \( S \), is denoted by

\[
\delta^+(S) \overset{\text{def}}{=} \{(u, v) \in E : u \in S, v \notin S\}
\]

The (directed) in-cut of \( S \), consisting of edges entering \( S \), is denoted by

\[
\delta^-(S) \overset{\text{def}}{=} \{(u, v) \in E : u \notin S, v \in S\}.
\]

### 10.1.7 Line embeddings and sweep cuts.

Recall that \( y \) also gives a set of edge lengths where the length of the shortest \((s, t)\)-path is 1. We leverage this insight to embed the vertices \( V \) on the real line — assigning values \( \alpha : V \to [0, +\infty) \) — as follows.

For each vertex \( v \), let \( \alpha_v \) be the length of the shortest \( s \leadsto v \) path w.r.t. the edge lengths \( y \in \mathbb{R}^n_\geq 0 \). We have \( \alpha_s = 0 \). We also have

\[
\alpha_t = \min_{p \in P_{s,t}} \sum_{e \in p} y_e \geq 1
\]

because of the covering constraints in \((\text{Min-Metric})\).

Consider the following random cut. We pick a value \( \theta \in (0, 1) \) uniformly at random. Let \( S = \{v : x_v \leq \theta\} \), and let \( \bar{S} = V \setminus S \). Since \( \alpha_s = 0 \) and \( \alpha_t \geq 1 \), the set of edges from \( S \) to \( \bar{S} \) is always an \((s, t)\)-cut.
The value of this cut is randomized. Let us bound the cost of the directed cut from \( S \) to \( \bar{S} \), in expectation. We have

\[
E \left[ \sum_{e \in \delta^{+}(S)} c(e) \right] = \sum_{e \in E} c(e) P\left[ e \in \delta^{+}(S) \right] \leq \sum_{e \in E} c(e) y_e = (\text{Min-Metric}) \quad (10.3)
\]

Here (a) is by linearity of expectation. The key inequality, (b), is based on the shortest path metric.

For an edge \( e = \{u, v\} \), we have \( e \in \delta^{+}(S) \) iff \( \alpha_u \leq \theta \leq \alpha_v \), which happens with probability at most \( \alpha_v - \alpha_u \). Consider the shortest path from \( s \) to \( u \) w/r/t \( y \), which has length \( \alpha_u \). Concatenating the shortest path from \( s \) to \( u \) with the edge \( e \),

\[
\alpha_u \rightleftharpoons u \rightarrow v,
\]

gives a walk of length \( \alpha_u + y_e \), hence \( \alpha_v \leq \alpha_u + y_e \).

Consider now the inequality obtained in (10.3),

\[
E \left[ \sum_{e \in \delta^{+}(S)} c(e) \right] \leq (\text{Min-Metric}).
\]

We have generated a randomized (discrete) cut that is on average no worse than the minimum fractional minimum cut. By the probabilistic method, there exists a value \( \theta \) where the \( (s,t) \)-cut has value at most this average. If not, then the average would have to be higher. This establishes the existence of a minimum cut with cost equal to the \( (s,t) \)-minimum cost metric, and establishes the max-flow min-cut theorem. To extract the cut, one can simply scan \( \theta \) over the interval \( (0, 1) \) and check all \( n-1 \) possible cuts. (In fact any \( \theta \in (0,1) \) will work; see exercise 10.1.)

### 10.2 Sparsest cut

Let \( G \) be undirected, and let \( b : \binom{V}{2} \to \mathbb{R}_{\geq 0} \) be a set of nonnegative demands. Given a set \( S \), the sparsity of \( S \) is defined as the ratio

\[
\frac{c(\delta(S))}{\sum_{u \in S, v \notin S} b(u,v)}.
\]
Sparsest cut

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 \textit{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 \textit{balanced cut} problem, as we will discuss.)

To drive this point further, observe that

\[
\frac{1}{n} \cdot \frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}} \leq \frac{c(\delta(S))}{|S||\bar{S}|} \leq \frac{2}{n} \cdot \frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}}
\]

because \( n/2 < \max\{|S|, |\bar{S}|\} \leq n \). That is, up to a constant factor, we are trying to minimize the ratio

\[
\frac{c(\delta(S))}{\min\{|S|, |\bar{S}|\}}.
\]

Here we clearly see the expense of choosing a very small set \( S \).

In this chapter, we describe a very influential result of Leighton and Rao \[LR99\] that obtains a deterministic \( O(\log(n)) \) approximation ratio for the uniform sparsest cut problem. Their algorithm is based on applying region growing to the metric induced by the dual LP. Given our interest in randomized algorithms, we first present an alternative, randomized \( O(\log(n)) \) approximation algorithm for general demands based on \( \ell_1 \)-\textit{metric embeddings}. We will also discuss lower bounds and some applications of sparsest cut.

10.2.1 The LP

Leighton and Rao’s algorithm \[LR99\], as well as the randomized algorithm via metric embeddings, are both based on rounding an LP relaxation of the sparsest cut problem. However, since the sparsest cut optimizes a ratio, obtaining the linear relaxation is not as obvious. As a step in this general direction, consider the following (nonlinear) relaxation of the sparsest cut problem.
Compute a metric $d : V \times V \to \mathbb{R}_{\geq 0}$ minimizing the ratio

$$\frac{\sum_{e=\{u,v\}} c(e)d(u,v)}{\sum_{\{u,v\}} b(u,v)d(u,v)}.$$ 

Now, we can scale the distances up or down with no effect on the ratio. In particular, we can fix the denominator to be 1, which gives the following optimization problem which is a linear program.

*Find the minimum $c$-cost metric such that the $b$-weighted sum of distances is at least 1.*

That is:

$$\text{minimize } \sum_{e=(u,v) \in E} c(e)d(u,v)$$

over all metrics $d : V \times V \to \mathbb{R}_{\geq 0}$

s.t. $\sum_{\{s,t\}} b(s,t)d(s,t) \geq 1.$

(10.4)

### 10.2.2 The dual LP and concurrent flow

To obtain the dual, it is helpful to rewrite (10.4) as a pure covering problem. Recall the correspondence between metrics and edge lengths, via shortest path distances. Then (10.4) is the same as:

*Find the minimum cost edge lengths such that the $b$-weighted sum of shortest path distances is at least 1.*

To make this more explicit, for $s, t \in V$, let $\mathcal{P}_{s,t}$ denote the family of all $(s,t)$-paths. Let us define a path bundle as a collection of paths $P$ consisting of an $(s,t)$-path $P_{s,t} \in \mathcal{P}_{s,t}$ for every pair $(s,t)$. We let $\mathcal{P}_s \overset{\text{def}}{=} \prod_{\{s,t\}} \mathcal{P}_{s,t}$ denote the family of all path bundles. Then we can express the problem above as follows.

$$\text{minimize } \sum_{e \in E} c(e)y(e) \text{ over } y : E \to \mathbb{R}_{\geq 0}$$

s.t. $\sum_{\{s,t\}} b(s,t) \sum_{e \in P_{s,t}} y(e) \geq 1$ for all $P \in \mathcal{P}_s.$

(10.5)

We can separate this LP by computing the shortest $(s,t)$-path for every pair $(s,t)$, and verifying the covering constraint for this bundle of shortest paths.
10. $L_1$-metric embeddings and sparsest cut

10.3. Rounding via $L_1$-metric embeddings

(10.5) covers path bundles with edges; thus, the dual packing LP packs path bundles into edges.

$$\text{maximize } \sum_{P \in \mathcal{P}_*} x(P) \text{ over } x : \mathcal{P_*} \rightarrow \mathbb{R}_{\geq 0}$$

$$\text{s.t. } \sum_{P \in \mathcal{P}_*} x(P) \sum_{(s,t) : e \in P_{s,t}} b(s, t) \leq c(e) \text{ for all } e \in E.$$ (10.6)

In (10.6), each path bundle $P$ represents a choice of paths for every $(s, t)$-pair to concurrently route the demands $b$. So (10.6) is trying to concurrently route $b$ as much as possible subject to the capacity constraints. This problem is called concurrent flow or demand multicommodity flow.

10.3 Rounding via $L_1$-metric embeddings

We now analyze an approach to rounding the sparsest-cut metric based on randomized embeddings. This version readily generalizes to general demands $b : \binom{V}{2} \rightarrow \mathbb{R}_{\geq 0}$.

Let $d$ be any metric that is a feasible solution to LP (10.4). Our goal is to convert $d$ to a cut with sparsity comparable to the cost of $d$.

10.3.1 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 $f : V \rightarrow \mathbb{R}$ 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

$$(\text{sparsity of } d) = \frac{\sum_{(u,v) \in E} c(u, v)d(u, v)}{\sum_{(s,t)} d(s, t)b(s, t)} = \frac{\mathbb{E}\left[\sum_{e \in \delta(S)} c(e)\right]}{\mathbb{E}\left[\sum_{s \in S, t \notin S} b(s, t)\right]}.$$ (10.7)
10. $L_1$-metric embeddings and sparsest cut

10.3. Rounding via $L_1$-metric embeddings

Note that the RHS is not the expected sparsity of $S$. The expected sparsity of $S$ is the quantity

$$E[\text{sparsity of } S] = E \left[ \frac{\sum_{e \in \delta(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s, t)} \right],$$

which is not the same as the quantities in (10.7). This is in contrast to our proof of max-flow min-cut, where $S$ is a minimum $(s,t)$-cut on average, and the existence of a minimum $(s,t)$-cut follows immediately from the probabilistic method.

Still the probabilistic approach can be salvaged with a little more work. Observe that $S$ can only be one of $n - 1$ different sets $S_1, \ldots, S_{n-1}$ 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

$$(10.7) = \frac{\sum_{i=1}^{n-1} p_i \sum_{e \in \delta(S_i)} c(e)}{\sum_{i=1}^{n-1} p_i \sum_{s \in S_i, t \in \bar{S}_i} b(s, t)}.$$

Now we apply the following elementary fact. (The proof is left as exercise 10.3.)

**Lemma 10.4.** Let $a_1, \ldots, a_h, b_1, \ldots, b_h > 0$. Then

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i} \leq \max_i \frac{a_i}{b_i}.$$

It follows that for some $S_i$, the sparsity of $S_i$ is at most the sparsity of $d$. In conclusion, we have shown the following.

**Lemma 10.5.** Let $d : \binom{V}{2} \to \mathbb{R}$ be a metric induced by a line embedding. Then one can partition the vertices into two sets $(S, \bar{S})$ such that

$$\frac{\sum_{e \in \delta(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s, t)} \leq \frac{\sum_{(u,v) \in E} c(u,v)d(u,v)}{\sum_{(s,t) \in E} d(u,v)b(s,t)}.$$

**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 \( f : V \rightarrow \mathbb{R}^h \). We can think of this as the sum of \( h \) line metrics \( f_1, \ldots, f_h \). The sparsity of \( d \) expands out to

\[
\text{(sparsity of } d \text{)} = \frac{\sum_{i=1}^{h} \sum_{\{u,v\} \in E} c(e) |f_i(u) - f_i(v)|}{\sum_{i=1}^{h} \sum_{\{u,v\} \in E} b(u,v)|f_i(u) - f_i(v)|}.
\]

Applying lemma 10.4 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 10.6.** 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

\[
\frac{\sum_{e \in \delta(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s,t)} \leq \frac{\sum_{\{u,v\} \in E} c(u,v)d(u,v)}{\sum_{(s,t) \in E} d(u,v)b(s,t)}.
\]

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.

### 10.3.2 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 10.6 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 10.7.** 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

\[
P[\|f(u) - f(v)\|_1 \leq d(u,v)] \leq \delta
\] (10.8)
For $\delta = 1/\text{poly}(n)$ and $h = O\left(\log^2 n\right)$, we can apply the union bound to (10.8) over all pairs $u, v$. This gives the following theorem.

**Corollary 10.8.** 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 10.8 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 (10.4) to obtain a sparsest metric $d$.
2. Invoke corollary 10.8 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 10.6 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 (10.4)).

This algorithm is due to Linial, London, and Rabinovich [LLR95], and establishes the following.

**Theorem 10.9.** There is a $O(\log n)$ randomized approximation to (non-uniform) sparsest cut (on undirected graphs).

Actually, Linial, London, and Rabinovich [LLR95] observed that one can do slightly better when there are demands for only $k$ commodity pairs. (i.e., $k$ pairs $(u, v)$ with $b(u, v) > 0$.)

**Theorem 10.10.** 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 10.4.

Now we describe the randomized algorithm of Linial, London, and Rabinovich [LLR95] that computes the embedding in theorem 10.14. We note that previously Bourgain [Bou85] had obtained a deterministic embedding but the output dimension
10. $L_1$-metric embeddings and sparsest cut

10.3. Rounding via $L_1$-metric embeddings

**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 $f : V \to \mathbb{R}^{\lceil \log n \rceil}$

Figure 10.1: A $O(\log n)$ dimension, randomized Frechét embedding with $O(\log n)$ distortion in expectation

Figure 10.2: Level sets by distance from a set of points $S$, encoding one coordinate of a Frechét embedding.

$h$ was exponential. Linial, London, and Rabinovich’s algorithm [LLR95] can be interpreted as an efficient randomized implementation of Bourgain’s embedding [Bou85].

The algorithm, which we call **random-Fréchet**, is extremely simple. We generate $\lceil \log n \rceil$ coordinates. For $i = 1, \ldots, n$, we sample a set $S_i$ where each point is sampled independently with probability $1/2^i$. For each vertex $v$, we find the distance between $v$ and (the closest point in) $S_i$. This gives the $i$th coordinate of $v$. Pseudocode is described in fig. 10.1.

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. 10.2 attempts to visualize a single coordinate generated in this manner.
10.3.3 Low-distortion in expectation

We now turn to proving theorem 10.14. Consider an instance of the random-Frechet algorithm, which computes a randomized embedding $f : V \rightarrow \mathbb{R}^{O(\log(n))}$.

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 10.11.** For $u, v \in V$ and $i \in \mathbb{N}$, $|u_i - v_i| \leq d(u, v)$.

**Proof.** By the triangle inequality, we have both

$$d(s, u) - d(s, v) \leq d(u, v) \quad \text{and} \quad d(s, v) - d(s, u) \leq d(u, v)$$

for all $s \in S_i$. \hfill \square

**Lemma 10.11** 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 10.12.** 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.,

$$r_i = \arg \min_{r > 0} \left\{ \left| \{x : d(u, x) \leq r\} \right| \geq 2^i, \left| \{x : d(v, x) \leq r\} \right| \geq 2^i \right\}$$

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 $c > 0$. 

172
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

$$
E[\|u - v\|_1] \geq \sum_{i=0}^{k} E[|u_{i+1} - v_{i+1}|] \\
\geq (a) \sum_{i=0}^{k} \left( \min\{r_i, \delta/2\} - \min\{r_{i-1}, \delta/2\} \right) \\
\geq (b) c\delta/4,
$$

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 10.13.** randomized-Fréchet embeds \( V \) into \( \mathbb{R}^{\lceil \log n \rceil} \) such that

\[
\|u - v\|_1 \leq \lceil \log n \rceil d(u,v) \text{ and } \mathbb{E}[\|u - v\|_1] \geq c d(u,v) \text{ for all } u,v \in V,
\]

for some absolute constant \( c > 0 \).

**10.3.4 Amplification**

Theorem 10.13 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 10.14.** 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

\[
c d(u,v) \leq \|u - v\|_1 \leq C \log(n) d(u,v) \text{ for all } u,v \in V
\]

for absolute constants \( c,C > 0 \).

**Proof sketch.** Fix \( u,v \in V \). We treat each coordinate difference \( |u_i - v_i| \) (for \( O(\log^2 n) \) coordinates over \( O(\log n) \) independent calls to random-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. \( \square \)
10.4 Rounding via region growing (for uniform demands)

Before the randomized approach of section 10.3 was discovered, Leighton and Rao [LR99] 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

$$c(\delta(S)) \frac{|S|}{|S||\bar{S}|}.$$ 

Let $d$ be an optimum metric to the metric relaxation (10.4). 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 $S$ with

$$c(\delta(S)) \frac{|S|}{|S||\bar{S}|} \leq O(\log(n))\text{OPT}_{(10.4)}.$$ 

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 $r_0 = 1/2n^2$.

10.4.1 Concentrated case.

First we consider the concentrated case.

**Lemma 10.15.** 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 $O(1)\text{OPT}_L$.

**Proof.** Suppose not. Then for all $q > 0$, we have

$$c(\delta(B(s, r_0 + q))) \geq c\text{OPT}_L |B(s, r_0 + q)||V \setminus B(s, r_0 + q)| \geq \frac{cn\text{OPT}_L}{3} |V \setminus B(s, r_0 + q)|$$

for a fixed universal constant $c > 0$ (to be determined). Here (a) observes that $|B(s, r_0 + q)| \geq |B(s, r_0)| \geq 2n/3$. 

175
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 $\text{OPT}_L$ (by a similar calculation as in multicut). Now, we have

$$1 \geq \frac{W(1) - W(r_0)}{\text{OPT}_L} = \int_0^{1-r_0} \frac{c(\delta(B(s, r_0 + q)))}{\text{OPT}_L} dq$$

$$\geq \frac{cn}{3} \int_0^{1-r_0} |V \setminus B(s, r_0 + q)| dq \overset{(b)}{=} \frac{cn}{3} \left( \sum_{v \in V \setminus B(s, r_0)} d(v, s) - r_0 \right)$$

$$\geq \frac{cn}{3} \sum_{v \in V} (d(v, s) - r_0) = \frac{cn}{3} \sum_{v \in V} d(v, s) - \frac{cnr_0}{3}. \quad (10.9)$$

(b) implicitly interchanges changes. In (c), the inequality holds because the sum is extended only by nonpositive terms.

We also have

$$1 \overset{(d)}{\leq} \sum_{\{u, v\}} d(u, v) \overset{(e)}{\leq} \sum_{\{u, v\}} d(u, s) + d(s, v) < n \sum_v d(v, s),$$

where (d) is by the distance constraint in the LP and (e) is by the triangle inequality.

Plugging back into (10.9), we have $1 > c/6$. For $c \geq 6$, then, we have a contradiction. \hfill \square

### 10.4.2 Non-concentrated case.

Now we address the remaining non-concentrated case.

**Lemma 10.16.** 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. $c(\cup_i \delta(S_i)) \leq O(n^2 \log n)\text{OPT}_L.$

2. $|S_i| \leq 2n/3$ for all $i$. 

176
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

$$\frac{c(\delta(T))}{|T||\bar{T}|} \leq \frac{c(\bigcup_i \delta(S_i))}{(2n/3)(n/3)} \leq O(\log n)\text{OPT}_L.$$ 

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

$$W(0) = \text{OPT}_L/n,$$

$$W'(r) = c(\delta(B(s, r))).$$

$W(r)$ 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 $W'(0) \geq c_0 \log(n)n^2W(r)$ implies that

$$W(r_0) \geq c_0 n^2 \log(n) r_0 > (1 + 1/n)\text{OPT}_L,$$

a contradiction. Thus there exists a radius $r \leq r_0$ that, in particular, satisfies

$$c(\delta(B(s, r))) = W'(r) \leq O\left(n^2 \log(n)\right)W(r) \leq O\left(n^2 \log(n)\right)\left(\text{OPT}_L/n + \sum_{v \in B(s, r)} \sum_{e \in \delta(v)} y_e c(e)\right).$$

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

$$c(\delta(B(s, r))) \leq O\left(n^2 \log n\right)\left(\frac{\text{OPT}_L}{n} + \sum_{v \in B(s, r)} \sum_{e \in \delta(v)} c(e)y_e\right).$$
10. L₁-metric embeddings and sparsest cut

10.5. Application: Minimum bisection

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

\[
c\left(\bigcup_i \delta(S_i)\right) \leq \sum_i c(\delta(S_i)) \\
\leq O(n^2 \log n) \left(\text{OPT} + \sum_{i \in S_i} \sum_{e \in \delta(v)} c(e)y(e)\right) \\
= O(n^2 \log(n)) \left(\text{OPT} + \sum_{v \in V} \sum_{e \in \delta(v)} c(e)y(e)\right) \\
= O(n^2 \log(n)) \text{OPT},
\]

which completes the proof. \qed

10.4.3 Putting it all together.

**Theorem 10.17** (Leighton and Rao [LR99]). 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 (10.5). 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. \qed

10.5 Application: Minimum bisection

A bisection is a partition of the vertices $V$ into $(S, \overline{S})$ of (essentially) equal size: $[n/2] \leq |S|, |\overline{S}| \leq [n/2]$. 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 $(S, \overline{S})$ – 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 10.5 guides the reader through a formal proof of the argument.

**Theorem 10.18.** 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.

## 10.6 Additional notes and materials

One can do better than a \( O(\log n) \) approximation for uniform sparsest cut – Arora, Rao, and Vazirani [ARV09] gave a \( O(\sqrt{\log n}) \) 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 [Rot16].
We refer to Spielman [Spi19] and Trevisan [Tre16] for further background on expanders, especially in the context of spectral graph theory. ?? follows the proof in [Tre16].

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

10.7 Exercises

Exercise 10.1. Recall the randomized rounding based proof of the max-flow min-cut theorem. Recall that we analyzed a random cut which was based on a threshold \( \theta \in (0, 1) \) chosen uniformly at random. Prove that for all \( \theta \in (0, 1) \), the corresponding cut is a minimum \( \{s, t\} \)-cut.

Exercise 10.2. In most textbooks, max flow is presented as the following LP, which in particular has polynomial size in the input graph \( G \).

\[
\begin{align*}
\text{maximize} & \quad \sum_{e \in \delta^+(s)} z_e - \sum_{e \in \delta^-(s)} z_e \\
\text{subject to} & \quad z_e \leq c(e) \quad \text{for all } e \in E \\
& \quad \sum_{e \in \delta^+(v)} z_e = \sum_{e \in \delta^-(v)} z_e \quad \text{for all } v \in V \setminus \{s, t\}.
\end{align*}
\]

The second set of constraints are called flow conservation constraints. Show that the above LP is equivalent to the (fractional path packing version of) \((\text{Max-Flow})\) in the following sense.

1. Show that for every (feasible) fractional path packing \( P \), there is a feasible solution \( z \) to (10.10) with the same objective value.

2. Show that for every feasible solution \( z \) to (10.10), there is a feasible path packing \( P \) with the same objective value.\(^8\)

Exercise 10.3. Prove lemma 10.4.

\(^8\)The second problem is trickier than the first. One should be able to prove it using only the ideas and results in this chapter (without retracing the flow algorithms of ensuing chapters).
Exercise 10.4. Recall the randomized \(O(\log n)\) approximation algorithm for sparsest cut based on \(L_1\)-embeddings. Note that this is also logarithmic in the number of demand pairs, \(\binom{n}{2}\). We consider the case where the demands are sparse; more precisely, where there are at most \(k\) commodities (i.e., pairs) \(\{s, t\}\) with nonzero demand \(b(s, t) > 0\).

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 10.5. Recall the bicriteria approximation algorithm for the minimum bisection problem from section 10.5.

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 \(O(\log n)\text{OPT}\).

Exercise 10.6. 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
\[
c(\delta^+(S))
\]
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 and Rao’s algorithm [LR99] to obtain a \(O(\log n)\) approximation for the directed uniform sparsest cut\(^9\).

Exercise 10.7. 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 10.6). 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.

\(^9\)This is not an easy exercise.
Exercise 10.8. Prove that any graph $G$ with constant expansion has diameter $O(\log n)$. 
Chapter 11

Tree Metrics

11.1 Introduction

Many flow and cut problems are ultimately about paths, whether packing paths in flow, cutting paths in cuts, evaluating shortest paths in fractional cuts, etc. Obviously there are many paths between any two points in a graph and this makes all these problems nontrivial. An extremely simple setting, then, would be a graph where there is a unique path between any two vertices: by definition, a tree. Many graphs problems become trivial in a tree. Here we will study a bold approach to graph algorithms based on this idea: process the input graph $G$ to produce a tree $T$ that preserves its salient properties, solve the problem on $T$, and lift the solution back to $G$. Of course a tree $T$ cannot preserve all of $G$, so we will only preserve specific properties and only approximately at that, in such a way that is appropriate to the problem at hand.

In this discussion, we will focus on preserving the shortest path metric of a graph. Let $G$ be an undirected graph, and let $d_G$ denote the shortest path metric in $G$. Let $T$ be a spanning tree of $T$ (with the same edge weights), and let $d_T$ denote the shortest path metric in $T$. (Of course, the shortest path in $T$ is also the only path.) Ideally, we want $d_T(u, v)$ to resemble $d_G(u, v)$ as much as possible for each edge $e = \{u, v\}$. In general, we have

$$d_G(u, v) \leq d_T(u, v)$$

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 $\alpha$. 

183
A natural goal is to obtain a spanning tree with small uniform stretch. However the \( n \)-vertex cycle \( C_n \) presents a lower bound of \( n - 1 \). Indeed, any spanning tree \( T \) of \( C_n \) is obtained by dropping one edge \( e \); this edge \( e \) is stretched around the cycle, so to speak, and has stretch \( n - 1 \).

The \( n \)-vertex cycle \( C_n \) indicates that we cannot, in the worst-case, find spanning trees with uniform stretch better than \( n - 1 \). (We leave it as exercise 11.1 to obtain a matching upper bound.) The rest of this chapter discusses two different approaches that obtain better bounds for relaxations of this problem. (In the lecture, we only discuss the second, randomized one in detail.)

**Low-stretch spanning trees.** Alon, Karp, Peleg, and West [AKPW95] showed how to compute a spanning tree \( T \) where the average stretch among all edges is \( n^{o(1)} \), in the sense that

\[
\frac{1}{|E|} \sum_{e \in E} (\text{stretch } e) \leq n^{o(1)}
\]

**Dominating tree metrics.** Bartal [Bar96; Bar98] ignored the requirement that \( T \) is a spanning tree of \( G \); more generally he sought auxiliary trees \( T \) where the 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 \( \text{polylog}(n) \):

\[
\mathbb{E}[\text{stretch } e] \leq O(\text{polylog}(n)) \text{ for all } e \in E.
\]

Note that this a different sense of “average stretch” then above. We will present an algorithm of [FRT04] building on [Bar98] to obtain (per-edge) average stretch of \( O(\log n) \).

### 11.2 Low-Stretch Spanning Trees

We first present the low-stretch spanning trees of [AKPW95]. Here we recall that we want to compute a spanning tree with low stretch on average over all the edges.

Suppose our goal was to obtain average stretch (roughly) \( D \) for a parameter \( D > 0 \). Let us partition the graph in vertex-disjoint subgraphs each with radius at most \( D/2 \) from some center vertex, and compute a shortest path tree from each center. Then every edge \textit{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
11. Tree Metrics

11.2. Low-Stretch Spanning Trees

[KAPW95] algorithm for $m^{o(1)}$-average stretch spanning trees:

1. Compute a low diameter decomposition: repeatedly, until no vertices remain:
   A. Select a remaining vertex $v$ and compute a shortest path tree of $v$ in the
      remaining graph.
   B. Remove the set $C_v$ all vertices (remaining) in $V$ within some distance
      $R' \leq R$ 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.

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
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

\[ (\#\text{ external edges}) \leq O(\log m) \frac{m}{R} \]

This revises the recursive bound as

\[ f(m) \leq Dm + (D + 1)f(c \log(m)m/D) \]
for a constant $c > 0$. For $D = e^{\sqrt{\log(m)/\log\log m}}$ the recursion is bounded by $f(m) = e^{O\left(\sqrt{\log m \log \log m}\right)}$. (The calculations are given below.)

**Theorem 11.1.** The AKPW algorithm returns a spanning tree with average stretch $e^{O\left(\sqrt{\log m \log \log m}\right)}$.

**Solving the recurrence.** We have

$$f(m) \leq Dm + (D + 1)f\left(c_0 \log(m)/D\right)$$

where $\epsilon = c_0 \log(m)/D$ for a constant $c_0$. The height of the recursion is

$$h = O\left(\log_D \log(m)\right) = O\left(\log(m)/\log(D/c_0 \log(m))\right) = O\left(\frac{\log(m)}{\log(D)}\right)$$

assuming $D = \Omega(\log m)$. Unrolling the recursion gives

$$f(m) \leq O(Dm) \sum_{i=0}^{h} \left((1 + 1/D)c_0 \log(m)\right)^i \leq Dme^{O(h \log \log m)}.$$

To minimize the RHS, we can instead minimize the logarithm of the RHS, $\log(m) + \log(D) + O(h \log \log m)$. Choosing $D$ to make the last two terms (roughly) equal, we have

$$\log(D) = \frac{\log(m) \log \log(m)}{\log(D)},$$

hence

$$\log(D) = \sqrt{\log(m) \log \log(m)}.$$

Then $D = e^{\sqrt{\log(m) \log \log(m)}}$ gives

$$f(m) \leq me^{O\left(\sqrt{\log(m) \log \log(m)}\right)},$$

as desired.
11.3 Hierarchical Tree Metrics

The [AKPW95] algorithm was able to obtain low stretch in total. This means that a few unlucky edges might have extremely high stretch. In this section we want every edge to have low stretch, in some sense. Unfortunately we already know that it is impossible to guarantee $o(n)$ stretch for every edge simultaneously. But a different possibility open up when we allow for randomization. Perhaps we can output a randomized tree $T$, such that for each edge $e$, the expected stretch of $e$ is $o(n)$. Such a claim does not contradict the lower bound for the $n$-vertex cycle; in fact, one can get constant expected stretch for the cycle which we leave as exercise 11.2.

To build some intuition, let us start from the [AKPW95] 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 [AKPW95].

1. We can make the radii of the clusters randomized, instead of a deterministic function of the total number of edges cut.

2. Second, the order of vertices that center the clusters can be randomized, which would seem most equitable.

Both of these ideas will be reflected at a high level in the following algorithm which we now present.

The algorithm we present will produce a randomized hierarchical tree metric over $V$. This means that the tree $T$ will be rooted, with $V$ at the leaves, and the edges between height $i$ and height $i-1$ have length $\alpha^i$ 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 constant factor of the biggest edges at the top of the corresponding subtree. This additional structure turns out to be useful for several other problems.

We now present [FRT04]’s randomized algorithm. We assume the input is an edge-weighted graph where the minimum edge length is normalized to 1. We let $D = \max_{u,v} d(u,v)$ denote the diameter of the graph. Below we describe the algorithm in detail and first we give a high level description. For every radius of the form $\alpha 4^i$, we are randomly scooping out balls of size $\alpha 4^i$, where $\alpha \in [1, 2]$ is drawn uniformly at random, and the vertices at the center of the balls are in random order. A key and subtle point is that we use the same (random) $\alpha$ and ordering for every $i$. the
intersections of these balls (across \(i\)'s) induce a laminar family of sets over \(V\) which are arranged as a tree.

\[\text{[FRT04]'s algorithm producing a randomized hierarchical tree metric.}\]

1. Let \(v_1, \ldots, v_n\) be a uniformly random ordering of \(V\). Let \(L = \lceil \log_4 D \rceil\). Let \(\alpha \in [1, 2]\) be drawn uniformly at random.
2. For \(i\) from \(L\) down to 0,
   (a) For each vertex \(v_j\) in order,
      i. Let \(C_{i,j}\) be the set of vertices at distance at most \(\alpha 4^{i-1}\) from \(v_j\), excluding any vertex already included by the cluster of a previous \(C_{i,j}\).
3. We use the \(C_{i,j}\)'s to arrange the vertices as leaves in a tree 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 \(C_{L,j}\).
   (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 11.2.** [FRT04]'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 \(d_T(u, v)\) is decided, up to a constant factor, by the height \(k\) where \(u\) and \(v\) are first separated. (Then \(d(u, v) = O(4^k)\).) 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

\[
d_T(u, v) \leq \sum_{w} O(1)(\text{contrib. of } w \text{ to } d_T(u, v)). \tag{11.1}
\]
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. (11.1) 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 11.3.** 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 assume without loss of generality that $w$ is closer to $u$ than $v$ (i.e., $d(u, w) \leq d(v, w)$). We first observe that $w$ 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
\[
d(w, u) \geq d(w, v) - d(u, v) > 4^k - 4^k/2 > 2 \cdot 4^{k-1}
\]
which rules out smaller values of $k$. Here (a) is by 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
\[
\mathbb{P}[E_1] \leq \frac{|[d(w, u), d(w, v)]|}{|4^k, 2 \cdot 4^k|} \leq \frac{d(u, v)}{4^k}
\]
by (c) the triangle inequality. Thus
\[
\mathbb{E}[\text{contrib. from } w \ldots] \leq 4^{k+1} \mathbb{P}[E_1] \mathbb{P}[E_2] \leq 4d(u, v)/\ell,
\]
as desired.

### 11.4 Additional notes and materials

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

### 11.5 Exercises

**Exercise 11.1.** Design and analyze an algorithm that computes a spanning tree with uniform stretch $n - 1$ (matching the lower bound induced by the cycle).

**Exercise 11.2.** For the $n$-vertex cycle $C_n$, describe a randomized tree metric where each edge has expected stretch $O(1)$.

**Exercise 11.3.** Recall that the low-stretch spanning tree of [AKPW95] obtained average stretch $n^{o(1)}$. We consider extensions to the weighted average. Let $w(e) \in \mathbb{R}_{>0}$ be a positive weight for every edge, and let $W = \sum_{e \in E} w(e)$ be the total weight. We
assume for simplicity that the weights are between 1 and poly(n). We still treat the edges as unit length edges. Design and analyze an 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 11.4.** Show how to use the randomized tree metric to randomly round the sparsest cut LP and obtain $O(\log n)$-approximation for sparsest cut.\(^1\)

**Exercise 11.5.** Prove that the $O(\log n)$ bound is tight for tree metrics (up to constants).

\(^1\)Try to prove tree metrics are also $L_1$-metrics.
Chapter 12

Sampling geometric range spaces

12.1 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. \((P, R)\) need not be distinct or finite. For example, \(P\) may be a collection of \(n\) points in \(\mathbb{R}^2\), and \(R\) may be the family of all closed discs in \(\mathbb{R}^2\). Typical queries, for a given \(r \in R\), include:

1. Is \(r\) empty?
2. How many points does \(r\) contain?
3. Does \(r\) contain at least an \(\epsilon\)-fraction of \(P\), for given \(\epsilon\)?

Extensions of this model include weighted points and ranges, or a distribution over points instead of a finite set. We focus on the unweighted and discrete setting for simplicity.

For a fixed range space \((P, R)\) with \(P\) finite, we define the measure \(\mu(r)\) of a range \(r \in R\) as the fraction of all the points it contains:

\[
\mu(r) = \frac{|r \cap P|}{|P|}.
\]

One can interpret \(\mu(r)\) as the probability that a random point from \(P\) 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,

$$|\mu(r) - \mu_Q(r)| \leq \epsilon$$

for all $r \in R$.

$Q$ is an $\epsilon$-net if it hits all ranges with measure at least $\epsilon$:

$$\mu_Q(r) > 0$$

for all $r \in R$ 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 $Q \subseteq P$ with $|Q| < (1 - \epsilon)|P|$, $Q$ cannot be an $\epsilon$-sample because in particular it will fail for the range $r = P \setminus Q$. For the same reason, any $\epsilon$-net $Q$ of $P$ must have at least $(1 - \epsilon)|P|$ points.

For a subset of points $Q \subseteq P$, let

$$R \land Q = \{ r \cap Q : r \in R \}$$

denote the family of subsets of $Q$ induced by $R$. For many natural range spaces, especially in low-dimensional geometry, $R \land Q$ is a very small subset of $2^Q$. Above we mentioned the setting of disks in the plane. The intersection of a disk with the point set $Q$ gives a subset of $Q$, but in general, disks cannot induce all possible subsets of a set of points. In fact, consider 4 points arranged in a square. It is impossible to take a disk and overlay it so that it only covers two opposite corners of the square. Thus disks are inherently of limited complexity, and we hope to leverage this when sampling.

One way to model this complexity is via the growth function. The growth function $g : \mathbb{N} \to \mathbb{N}$ models the maximum cardinality of $R \land Q$ as a function of $|Q|$. For $k \in \mathbb{N}$, $g(k)$ is defined by

$$g(k) \overset{\text{def}}{=} \max\{|R \land Q| : Q \subseteq P, |Q| \leq k\}.$$ 

$(P, R)$ is said to have polynomial growth of degree $d$ if

$$g(k) \leq O(k^d).$$

For example, disks in the plane have polynomial growth of degree 3 (as we prove in section 12.3). The family of closed halfspaces in $\mathbb{R}^d$ have polynomial growth of degree
There are several ways to establish whether a range space has polynomial growth, via parameters such as the VC-dimension (section 12.3) and the shattering dimension (see [Har18b]).

Now, let \((P, R)\) be a range space with growth function \(g(n)\). Let \(\epsilon, \delta \in (0, 1)\), and let \(Q \subset P\) be a random sample of \(P\). We want to understand how big \(Q\) should be so that it is either an \(\epsilon\)-sample or an \(\epsilon\)-net with probability at least \(1 - \delta\).

A straightforward approach to this question is to apply the following additive Chernoff bound.

**Theorem 12.1.** Let \(X_1, \ldots, X_n \in [0, 1]\) be independent, \(\mu = \mathbb{E}[X_1 + \cdots + X_n]/n\) and \(\epsilon > 0\). Then

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

The proof, which is similar to that of the multiplicative Chernoff bound, is deferred to section 12.A.

Suppose we want an \(\epsilon\)-sample for the range space \((R, P)\) with growth function \(g(n)\). Let \(Q \subset P\) be a random sample (with repetition) of \(k\) points and let \(\mu_Q\) be the corresponding measure. Fix a range \(r\) and consider \(\mu_Q(r)\). We have \(\mathbb{E}[\mu_Q(r)] = \mu(r)\). Write

\[\mu_Q(r) = \frac{1}{k}(X_1 + \cdots + X_k)\]

where \(X_i = 1\) if the \(i\)th sampled point lies in \(r\), and 0 otherwise. Thus

\[\mathbb{P}[|\mu_Q(r) - \mu(r)| \geq \epsilon] = \mathbb{P}\left[\left|\frac{1}{k}(X_1 + \cdots + X_k) - \mu(r)\right| \geq \epsilon\right] \leq 2e^{-\epsilon^2 k}.
\]

Now, there are effectively at most \(|R \wedge P| \leq g(|P|)\) distinct ranges we want to preserve. For a given parameter \(\delta \in (0, 1)\), and \(k = \log(g(|P|)/2\delta)/\epsilon^2\) we have

\[\mathbb{P}[Q \text{ is not an } \epsilon\text{-sample}] \leq \sum_{r \in R \wedge P} \mathbb{P}[|\mu_Q(r) - \mu(r)| \geq \epsilon] \leq |R \wedge P| \cdot \frac{\delta}{g(|P|)} \leq \delta\]

by (a) the union bound. That is, \(\log(g(|P|)/2\delta)/\epsilon^2\) points suffice to give an \(\epsilon\)-sample with probability at least \(1 - \delta\). For example, for disks in the plane, we need to sample \(O(\log(|P|)/2\epsilon^2)\) points to obtain an \(\epsilon\)-sample with constant probability.

Meanwhile, for \(\epsilon\)-nets, one can show a random sample of size \(O(\log(g(|P|))/\epsilon)\) is an \(\epsilon\)-net with constant probability in a relatively straightforward fashion. (See exercise 12.1.)
12. Sampling geometric range spaces

12.2. Proof of the \( \epsilon \)-sample theorem

It turns out that one can do much better than these initial bounds, and surprisingly, remove the dependence on \(|P|\) entirely. These stronger sampling bounds, called the \( \epsilon \)-sample theorem and \( \epsilon \)-net, are the main topic of this chapter. For \( \epsilon \)-samples we have the following [VC71].

**Theorem 12.2.** Let \((P,R)\) be a range space with growth function \(g(n)\), and let \( \epsilon, \delta \in (0, 1) \) be given. Let \( \ell \in \mathbb{N} \) such that

\[
\ell \geq C \log\left(\frac{g(2\ell)/\delta}{\epsilon^2}\right)
\]

for a universal constant \(C\). Then a random sample of \( \ell \) points with repetition from \( P \) is an \( \epsilon \)-sample with probability at least \( 1 - \delta \).

We prove theorem 12.2 in section 12.2. Note that \(|Q|\) is independent of \(|P|\). For \( \epsilon \)-nets an even smaller sample suffices [HW87]:

**Theorem 12.3.** Let \((P,R)\) be a range space with growth function \(g(n)\), and let \( \epsilon, \delta \in (0, 1) \) be given. Let \( \ell \in \mathbb{N} \) such that

\[
\ell \geq C \log\left(\frac{g(2\ell)/\delta}{\epsilon}\right)
\]

for a universal constant \(C\). Then a random sample of \( \ell \) points with repetition from \( P \) is an \( \epsilon \)-sample with probability at least \( 1 - \delta \).

The proof of theorem 12.3 is similar to the proof of theorem 12.2, and left to the reader in exercise 12.2.

We mention that the theorems above can be formulated in a continuous setting, where instead of a finite point set \(P\), we have a distribution \(D\) of points. Here the measure \(\mu(r)\) of a range \(r\) is the probability of a point drawn from \(D\) lying in \(r\). The proofs of the \( \epsilon \)-sample and \( \epsilon \)-net theorems presented here extend immediately to these settings. We focus on the discrete setting as it captures the essential ideas while being simpler to discuss.

### 12.2 Proof of the \( \epsilon \)-sample theorem

In the section we prove the \( \epsilon \)-sample theorem. We first restate the theorem for the reader’s convenience.

**Theorem 12.2.** Let \((P,R)\) be a range space with growth function \(g(n)\), and let \( \epsilon, \delta \in (0, 1) \) be given. Let \( \ell \in \mathbb{N} \) such that

\[
\ell \geq C \log\left(\frac{g(2\ell)/\delta}{\epsilon^2}\right)
\]

for a universal constant \(C\). Then a random sample of \( \ell \) points with repetition from \( P \) is an \( \epsilon \)-sample with probability at least \( 1 - \delta \).
Proof. Let \( A \) be the event that \( Q_1 \) is incorrect for some \( r \in R \).

\[ A \overset{\text{def}}{=} \text{the event that} \ |\mu_1(r) - \mu(r)| > \epsilon \text{ for some } r. \]

We want to show that \( P[A] < \delta \).

Let \( Q_2 \) be a second, independent random sample of the same size. Define \( B \) by

\[ B \overset{\text{def}}{=} \text{the event that} \ |\mu_2(r) - \mu_1(r)| > \epsilon/2 \text{ for some } r \in R. \]

Define \( C \) as the event where the conditions of events \( A \) and \( B \) hold simultaneously for the same \( r \in R \). That is, let \( C \) be the event that \( |\mu_1(r) - \mu(r)| > \epsilon \text{ and } |\mu_1(r) - \mu_2(r)| > \epsilon/2 \) for some \( r \in R \).

We claim that \( P[A] \leq 2P[B] \). To this end, we first write

\[ P[B] \geq P[C] = P[A, C] = P[C \mid A] P[A]. \tag{12.1} \]

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 \( 1/2 \) by the additive Chernoff inequality. In this case we have

\[ |\mu_1(r) - \mu_2(r)| \geq |\mu(r) - \mu_1(r)| - |\mu(r) - \mu_2(r)| > \epsilon - \epsilon/2 = \epsilon/2, \]

hence event \( C \). Thus \( P[C \mid A] \geq 1/2 \), which gives \( P[B] \geq P[A]/2 \) when plugged into (12.1).

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 \( 2k \) points \( Q_0 \) from \( P \).
2. Randomly partition \( Q_0 \) in half. Let \( Q_1 \) be one half and let \( Q_2 \) be the other.

Since

\[ P[B] = \sum_{Q_0} P[Q_0] P[B \mid Q_0] \leq \max_{Q_0} P[B \mid Q_0], \]

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

\[ P[|\mu_1(r) - \mu_0(r)| \geq \epsilon/4] \leq \frac{\delta}{4m} \]

196
as well as
\[
P[|\mu_2(r) - \mu_0(r)| \geq \epsilon/4] \leq \frac{\delta}{4m}
\]
by the additive Chernoff inequality. By taking the union bound over all \(m\) distinct ranges over \(Q_0\) we have \(|\mu_1(r) - \mu_0(r)| \leq \epsilon/4\) and \(|\mu_2(r) - \mu_0(r)| \leq \epsilon/4\) for all \(r \in R\) with combined probability of error at most \(\delta/2\). In this case we have
\[
|\mu_1(r) - \mu_2(r)| \leq |\mu_1(r) - \mu_0(r)| + |\mu_2(r) - \mu_0(r)| = \epsilon/2
\]
for all \(r \in R\); i.e., event \(B\) does not occur. Thus \(P[B] \leq \delta/2\), completing the proof.

### 12.3 VC dimension

We have shown that very few points are needed for \(\epsilon\)-samples and \(\epsilon\)-nets when the range system \((P, R)\) has small (e.g., polynomial) growth \(g(n)\). But how do we bound the growth function to begin with? One way to do this is via a property called the VC dimension of \((P, R)\), and this is the topic of this section.

For \(Q \subset P\), we say that \(R\) shatters \(Q\) if the family of projections \(R \land Q\) induces all \(2^{|Q|}\) subsets of \(Q\).

The VC dimension of \((P, R)\) is the maximum cardinality \(|Q|\) of any shattered subset \(Q \subset P\).

#### Intervals

Let \(P\) be the real line \(\mathbb{R}\), and let \(R\) be the family of intervals on \(\mathbb{R}\). Any two points can be shattered by intervals, but given any three points \(a < b < c\), it is impossible to induce the set \(\{a, b\}\) with an interval.

#### Disks

Let \(P\) be the plane \(\mathbb{R}^2\), and let \(R\) be the family of closed disks. Any three points can be shattered by disks. Consider a set \(Q\) of four points. We have two cases.

In one case, one of the four points is contained in the convex hull of the others. Then it is impossible to find a disk that covers the three outer points without including the point in their convex hull, since disks are convex. (In general, any set of points that are not in convex position cannot be shattered by range spaces of convex shapes.)

Now suppose all four points are in convex position. Call these points \(\{a, b, c, d\}\) in order along the convex hull. If we had two disks \(D_1\) and \(D_2\) that induced the “opposite” pairs \(\{a, c\}\) and \(\{b, d\}\), respectively, then the boundaries of \(D_1\) and \(D_2\) would intersect at four points. But two circles can only intersect at two points.

#### Halfspaces

Let \(P = \mathbb{R}^d\), and let \(R\) be the set of closed halfspaces:
\[
R = \left\{ \{x \in \mathbb{R}^d : \langle a, x \rangle \leq b\} : a \in \mathbb{R}^d, b \in \mathbb{R} \right\}.
\]

Here we have the following fact.
Theorem 12.4 (Radon’s theorem). Let $Q$ be a set of $d+2$ points in $\mathbb{R}^d$. Then one can partition $Q$ into two sets $S_1 \cup S_2$, such that $\text{conv}(S_1) \cap \text{conv}(S_2) \neq \emptyset$.

Radon’s theorem implies that the VC-dimension is at most $d+1$. Indeed, if $Q$ had $d+2$ points, then partition $Q = S_1 \cup S_2$ as in Radon’s theorem. If a halfspace $H$ contains $S_1$, then it also contains a point in the convex hull of $S_2$. But then it $H$ must also contain at least one point from $S_2$. Thus it is impossible to have $H \cap Q = S_1$. To show that VC-dimension is exactly $d+1$, one verifies that the regular simplex in $\mathbb{R}^d$ with $d+1$ vertices can be shattered by half spaces.

12.3.1 Sauer’s lemma

Theorem 12.5. Let $(P, R)$ be a range space of VC-dimension $d_{vc}$. Then $(P, R)$ has growth function

$$g(n) \leq \sum_{i=0}^{d_{vc}} \binom{n}{i} \leq n^{d_{vc}}.$$

Proof. Define $f(d_{vc}, n) = \sum_{i=0}^{d_{vc}} \binom{n}{i}$. Clearly $g(n) \leq f(d_{vc}, n)$ when $n = 0$ or $d_{vc} = 0$.

Let $Q \subset P$ have $n$ points and let $p \in Q$. Without loss of generality we may assume that each range is a subset of $Q$, that is,

$$r \subseteq Q \text{ for all } r \in R.$$

(Otherwise replace $R$ with $R \land Q$.) Define two subsets of ranges by:

- $R_p \overset{\text{def}}{=} \{ r - p : r \in R, r + p \in R, r - p \in R \}$
- $R^p \overset{\text{def}}{=} \{ r - p : r \in R \}$.

Here we adopt the shorthand $r + p \overset{\text{def}}{=} r \cup \{p\}$ and $r - p = r \setminus \{p\}$. We claim that $|R| = |R_p| + |R^p|$. To see this, observe that for a range $r \in R$, we have one of three cases:

1. $p \notin r$ and $r \cup \{p\} \notin R$, in which case $r$ is in $R^p$.
2. $p \notin r$ and $r \cup \{p\} \in R$, in which case $r \in R_p$ and $r \in R^p$.
3. $p \in r$ and $r \setminus \{p\} \in R$, in which case $r \notin R_p$ and $r \in R^p$. 

198
Note that for each range $r$ of the second type, which is double-counted, the range $r \cup \{p\}$ appears in the third type which is omitted. (And vice versa.) This proves the claim.

The range space $(Q - p, R_p)$ has VC-dimension at most $d_{vc} - 1$, since if $S \subset P$ can be shattered by $R_p$, then $S \cup \{p\}$ can be shattered by $R$. Meanwhile, the range space $(Q - p, R^p)$ has VC-dimension at most $d_{vc}$. By induction, we have

$$|R| = |R_p| + |R^p| \leq f(d_{vc} - 1, n - 1) + f(d_{vc}, n - 1).$$

Moreover,

$$f(d_{vc} - 1, n - 1) + f(d_{vc}, n - 1) = \sum_{i=0}^{d_{vc}-1} \binom{n-1}{i} + \sum_{i=0}^{d_{vc}} \binom{n-1}{i}$$

$$\leq \sum_{i=0}^{d_{vc}} \binom{n-1}{i-1} + \sum_{i=0}^{d_{vc}} \binom{n-1}{i}$$

$$\leq \sum_{i=0}^{d_{vc}} \binom{n}{i} = f(d_{vc}, n),$$

as desired. To see (a), observe that the RHS counts the number of subsets of size $i$ from a universe of $n$ points. The LHS counts the same collection by counting the number of such sets that contains a given point in the first sum, and counting the number of such sets that omits a given point in the second sum.

12.4 Additional notes and materials

See [Har11; Har18b] for additional background on geometric sampling and complexity, including another notion of geometric complexity called the shattering dimension which is omitted here. [Har11; Har18b] also describes an interesting connection between geometric sampling and discrepancy.

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

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

12.5 Exercises

Exercise 12.1. Provide directly (and without leveraging the techniques from section 12.2) that sampling $O(\log(g(|P|))/\epsilon^2)$ points gives an $\epsilon$-net with constant probability.
Exercise 12.2. This exercise asks you to prove the $\epsilon$-net theorem, theorem 12.3. One can prove it similarly to the $\epsilon$-sample theorem, theorem 12.2. Below we define the events $A$ and $B$ for you to get you started, and ask you to complete the proof (in full detail).

Let $Q_1$ and $Q_2$ be two random samples of points (of appropriate size, TBD by you) inducing measures $\mu_1$ and $\mu_2$, respectively. Define the events $A$ and $B$ by:

$$A \overset{\text{def}}{=} \text{the event that } Q_1 \cap r = \emptyset \text{ and } \mu(r) \geq \epsilon \text{ for some } r \in R.$$  
$$B \overset{\text{def}}{=} \text{the event that } Q_1 \cap r = \emptyset \text{ and } \mu_2(r) > \epsilon/2 \text{ for some } r \in R.$$  

12.A Proof of the additive Chernoff bound

In this section we prove the additive Chernoff bound. First we required the following upper bound on the moment generating function of a bounded random variable with mean 0.

Lemma 12.6. Let $X \in [-1,1]$ be a bounded random variable with mean $\mathbb{E}[X] = 0$. Then for all $t \in \mathbb{R}$,

$$\mathbb{E}[e^{tx}] \leq e^{t^2/2}.$$  

Proof. Observe that for all $x \in [-1,1]$, we have

$$x = \frac{1 + x}{2} - \frac{1 - x}{2},$$

hence by convexity of $f(x) = e^{tx}$, we have

$$e^{tx} \leq \left(\frac{1 - x}{2}\right)e^{-t} + \left(\frac{1 + x}{2}\right)e^t.$$  

Consequently

$$\mathbb{E}[e^{tx}] \leq \mathbb{E}\left[\left(\frac{1 - X}{2}\right)e^{-t} + \left(\frac{1 + X}{2}\right)e^t\right] = \frac{e^t + e^{-t}}{2}.$$  

200
Moreover, by the Taylor expansion of $e^x$, we have
\[
\frac{1}{2}(e^t + e^{-t}) = \frac{1}{2}\left(\left(1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots\right) + \left(1 - 1 + \frac{t^2}{2} - \frac{t^3}{3!} + \cdots\right)\right)
\]
\[
= 1 + \frac{t^2}{2} + \frac{t^4}{4!} + \frac{t^6}{6!} + \cdots
\]
\[
\leq 1 + \frac{t^2}{2} + \frac{(t^2/2)^2}{2!} + \frac{(t^2/2)^3}{3!} + \cdots
\]
\[
= e^{t^2/2},
\]
as desired. \qed

We now prove the additive Chernoff bound, which we first restate for the reader’s convenience.

**Theorem 12.1.** Let $X_1, \ldots, X_n \in [0, 1]$ be independent, $\mu = \mathbb{E}[X_1 + \cdots + X_n]/n$ and $\epsilon > 0$. Then
\[
\mathbb{P}\left[\frac{1}{n}(X_1 + \cdots + X_n) \geq \mu + \epsilon\right] \leq e^{-2\epsilon^2 n},
\]
\[
\mathbb{P}\left[\frac{1}{n}(X_1 + \cdots + X_n) \leq \mu - \epsilon\right] \leq e^{-2\epsilon^2 n}.
\]

**Proof.** We only prove the first bound; the second bound follows from the first by considering the variables $Y_i = 1 - X_i$. We also prove the inequality for a weaker constant of $1/2$ in the exponent instead of $2$. Obtaining the factor of $2$ requires more calculations and we refer the reader to [Har18a] for this proof.

The high-level proof idea is similar to the multiplicative proof idea – we exponentiate and apply Markov’s inequality, and leverage independence to reduce the analysis to analyzing the moment generating function of each $X_i$ separately.

For each $X_i$, let $\mu_i = \mathbb{E}[X_i]$. Thus $n\mu = \mu_1 + \cdots + \mu_n$. Let $t \in (0, 1)$ be a parameter TBD. We have
\[
\mathbb{P}\left[\frac{1}{n}(X_1 + \cdots + X_n) \geq \mu + \epsilon\right] = \mathbb{P}[X_1 + \cdots + X_n \geq \mu_1 + \cdots + \mu_n + \epsilon n]
\]
\[
= \mathbb{P}\left[e^{t(X_1 + \cdots + X_n)} \geq e^{t(\mu_1 + \cdots + \mu_n + \epsilon n)}\right]
\]
\[
\leq \mathbb{E}\left[e^{t(X_1 + \cdots + X_n - \mu_1 - \cdots - \mu_n - \epsilon n)}\right]
\]
\[
\overset{(\approx)}{=} e^{-\epsilon tn} \prod_{i=1}^{n} \mathbb{E}\left[e^{t(X_i - \mu_i)}\right].
\]
(a) is by the independence of the $X_i$'s. For each $i$, $X_i - \mu_i$ is bounded in $[-1, 1]$ and has mean 0. By lemma 12.6, we have

$$\mathbb{E}[e^{t(X_i - \mu_i)}] \leq e^{t^2/2}.$$ 

Plugging back in, we have

$$P[\ldots] \leq e^{nt^2/2 - t\epsilon n}.$$ 

The RHS is minimized by setting $t = \epsilon$, hence

$$P[\ldots] \leq e^{\epsilon^2 n/2}.$$ 

$\square$
Chapter 13

PAC learning

In this chapter we discuss foundations of learning theory particularly as it relates to randomized analysis. We start from the consistent learning model, which is appealingly simple but does not capture generalization error. This leads to the probabilistic approximately correct (PAC) learning model, which crucially incorporates randomization to correct this deficiency. From there we explore the notion of generalization error and study its correspondence to concentration inequalities that we have already studied.

13.1 The consistency model

In this chapter we are interested classification algorithms that assign labels to unlabeled data, generally based on previous experience analyzing a limited set of labeled training data. Instead of the design and analysis of such algorithms, we focus on developing models that allow us to more sensibly declare if such an algorithm is really “learning” or not.

Our first model of learning is called the consistency model. Let $X$ denote the space of all possible examples, also called the domain. We consider binary classification problems where each $x \in X$ is labeled 0 or 1. A mapping $c : X \rightarrow \{0, 1\}$ from examples to labels is called a concept. We assume that $X$ is labeled by an unknown concept $c$ from a known collection of concepts $C \subseteq 2^X$, also called a concept class.

The high-level goal is to identify the underlying true concept $c$ given access to a limited set of data. More precisely, the input concepts of a set of labeled examples

$$(x_1, y_1), \ldots, (x_m, y_m)$$

where $x_i \in X$ and $y_i = c(x_i) \in \{0, 1\}$. A concept $c'$ is consistent if $c'(x_i) = y_i$ for all $i$. A learning algorithm in the consistency model is one that, given the labeled examples above, either

(a) outputs a concept $c' \in C$ consistent with the examples, or
13. PAC learning

13.1. The consistency model

Kent Quanrud
Fall 2022

(b) decides that no such concept exists.

The concept class $C$ is learnable in the consistency model if there is a learning algorithm in the consistency model for it.

Of course we are only interested in efficient learning algorithms. The concept class $C$ is typically extremely large or even infinite, so directly inspecting all concepts in $C$ is too slow.

It is obviously desirable to be able to identify a concept $c' \in C$ that agrees with all the labeled input data. However, the consistency model does not ask $c'$ to agree the true concept $c$ beyond the labeled data. This can lead to some undesirable behavior, such as overfitting, as in the following examples.

Example: disjunctions. Recall that a boolean formula $f(x_1,\ldots,x_n)$ is a disjunction if it is the disjunction ($\lor$) of $x_i$’s and $\bar{x}_i$’s. Let $X = \{0,1\}^n$, and let $C$ be the class of disjunctions over $n$-variables. That is, each $c \in C$ corresponds to a disjunction $f_c(x_1,\ldots,x_n) = x_{i_1} \lor \bar{x}_{i_2} \lor \cdots \lor \bar{x}_{i_k}$, where $c(x) = 1$ iff $f_c(x) = \text{true}$.

Given a set of labeled points $\{(x_i,y_i)\}$ in $\mathbb{R}^d$, we can define a disjunction $f(x_1,\ldots,x_n) = x_{j_1} \lor \cdots \lor x_{j_k}$ where we include:

(a) Variable $x_j$ iff $x_{i,j} = 0$ for all samples $x_i$ with $y_i = 0$.

(b) Negated variable $\bar{x}_j$ iff $x_{i,j} = 1$ for all samples $x_i$ with $y_i = 1$.

Observe that $f$ will correctly label all of the data in the sample, and the algorithm to construct $f$ is efficient. Thus this concept class is efficiently learnable in the consistency model.

In general, if $f^*(x_1,\ldots,x_n)$ is the disjunction defining the true concept $c^*$, then the function $f$ we construct is a “superset” of $f^*$ in terms of the symbols $x_i$ and $\bar{x}_i$ it contains. Therefore if $f$ is much larger than $f^*$, then it will overfit the training sample and generally label many more $x$’s as true than $f^*$.

Example: rectangles. Let $X = [0,1]^2$, and let $C$ be the set of concepts defined by rectangles in $[0,1]^2$. That is, each $c \in C$ is associated with a rectangle $R_c = [a,b] \times [c,d]$, and $c(x) = 1$ iff $x \in R_c$.

Given a set of labeled points $\{(x_i,y_i)\}$, it is easy to identify the smallest rectangle $R_1$ containing all (and only) the $x_i$’s labeled $y_i = 1$. Likewise one can identify the largest rectangle $R_2$ containing the same $x_i$’s. So by outputting either $R_1$ or $R_2$ (or anything in between), $(X,C)$ is efficiently learnable in the consistency model.

All we know about the true concept is that it is induced by a rectangle $R^*$ such that $R_1 \subseteq R^* \subseteq R_2$. If the difference between $R_1$ and $R_2$ is very big (particular when the data is limited and/or not uniformly spread), then neither $R_1$ nor $R_2$ may be a very good classifier on real data.
13.2 The PAC learning model

The problem with the consistency model is that it does not incorporate generalization beyond the given training examples. This weakness is address in our next model, called the probably approximately correct (PAC) learning model and introduced in [Val84a; Val84b].

The PAC learning model is an extension of the consistent learning model. We retain the same setup where \( X \) denotes the space of examples, \( \{0, 1\} \) the set of labels, and \( \mathcal{C} \) the concept class. The new ingredients are as follows. First, we assume that the training examples are independently and identically distributed from a fixed (but unknown) distribution \( D \). Second, we define a second family of labeling functions, \( \mathcal{H} \subseteq 2^X \), called the hypothesis class. (Possibly \( \mathcal{H} = \mathcal{C} \).) Our algorithms output a hypothesis \( h \in \mathcal{H} \) that, ideally, agrees as much as possible with the unknown concept \( c \). The classification error of a hypothesis \( h \) is measured by

\[
\text{err}(h) \overset{\text{def}}{=} \mathbb{P}_{x \sim D}[h(x) \neq c(x)].
\]

We now define a PAC learning algorithm for \((\mathcal{C}, D)\). Let \( \epsilon, \delta \in (0, 1) \) be fixed parameters. A PAC learning algorithm takes as input \( m = \text{poly}(1/\epsilon, 1/\delta) \) (labeled) samples drawn from \( D \), and produces a hypothesis \( h \). The output hypothesis \( h \) is randomized since it reflects a randomized training set; besides, the algorithm itself is allowed to use additional randomization. To meet the criteria of PAC learning, the hypothesis \( h \) must have classification error at most \( \epsilon \) with probability \( 1 - \delta \). This requirement can be stated compactly as:

\[
\mathbb{P}_h[\text{err}(h) \leq \epsilon] = \mathbb{P}_h[\mathbb{P}_{x \sim D}[h(x) \neq c(x)] \leq \epsilon] \geq 1 - \delta. \tag{13.1}
\]

(13.1) is a mathematical expression of generalization. Unlike the consistency model of learning, PAC learning does not require \( h \) to be particularly accurate on the sample set. (13.1) is only concerned with \( h \)'s performance with respect to the entire distribution \( D \).

**Example: rectangles.** Recall the earlier example where the concepts are induced by rectangles in \([0, 1]^2\). There we observed that we have efficient consistent learning algorithms by outputting either the smallest rectangle \( R_1 \) or largest rectangle \( R_2 \) containing all of the positively labeled examples. We also observed that the true rectangle \( R^* \) lied somewhere between \( R_1 \) and \( R_2 \), which is a weak guarantee if \( R_1 \) and \( R_2 \) differ greatly.

Suppose \( D \) drew points \( x \in [0, 1]^2 \) uniformly from \([0, 1] \). As the number of samples \( m \) grows large, we expect points that our closer and closer to each side of the boundary
of $R^*$, and $R_1$ and $R_2$ should converge over time. One can show that this concept is PAC learnable by directly calculating the generalization error for either the smallest or largest enclosing rectangle. We will prove this later (perhaps with weaker constants) when we more generally analyze concept classes based on their VC dimension.

### 13.3 Generalization for finite hypothesis classes

We first analyze the generalization error for a learning algorithm in the consistent learning model, as a function of the number of hypotheses, $|\mathcal{H}|$. In particular these bounds only hold when $\mathcal{H}$ is finite. In the following theorem, we need $m$ to be proportional to $\ln(|\mathcal{H}|/\delta)/\epsilon$ to guarantee low generalization error.

**Theorem 13.1.** Let $\epsilon, \delta \in (0, 1)$ and consider a training set of size $m$ drawn from $\mathcal{D}$. If $m \geq \ln(|\mathcal{H}|/\delta)/\epsilon$, then with probability at least $1 - \delta$, every consistent hypothesis $h \in \mathcal{H}$ has true error $\text{err}(h) \leq \epsilon$.

**Proof.** Call a hypothesis $h \in \mathcal{H}$ good if $\text{err}(h) \leq \epsilon$, and bad if $\text{err}(h) > \epsilon$. It suffices to show that with probability at least $1 - \delta$, all bad hypotheses are inconsistent with the training data.

Fix a bad hypothesis $h$. For each sample $(x_i, y_i) \sim \mathcal{D}$, we have $\Pr[h(x_i) = y_i] < 1 - \epsilon$. The probability that $h$ agrees on all $m$ independent samples $(x_1, y_1), \ldots, (x_m, y_m)$ is

$$(1 - \epsilon)^m \leq e^{-\epsilon m} \leq \frac{\delta}{|\mathcal{H}|}.$$ 

By the union bound, the probability that some bad hypothesis $h$ is consistent is at most

$$\sum_{h \text{ bad}} \Pr[h \text{ consistent}] \leq |\mathcal{H}| \cdot \frac{\delta}{|\mathcal{H}|} \leq \delta,$$

as desired. \(\square\)

**Uniform convergence.** For a training set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ and hypothesis $h$, let

$$\text{err}(h \mid S) = \frac{1}{m} |\{i : h(x_i) \neq y_i\}|$$

denote the empirical or training error rate of $h$ with respect to $S$. Most learning algorithms try to select $h \in \mathcal{H}$ minimizing the training error $\text{err}(h \mid S)$, in the hopes that it would minimize $\text{err}(h)$ as well. Here we run two kinds from risk when the empirical error $\text{err}(h \mid S)$ deviates from the true error $\text{err}(h)$. 

206
13. PAC learning

13.4 Occam’s razor

1. A very good hypothesis $h$ with low $\text{err}(h)$ may have $\text{err}(h|S)$ much larger than $\text{err}(h)$, dissuading the algorithm from selecting $h$.

2. A very bad hypothesis $h$ with high $\text{err}(h)$ may have $\text{err}(h|S)$ much smaller than $\text{err}(h)$, tempting the algorithm from selecting $h$.

Thus, for minimizing $\text{err}(h|S)$ to be a good strategy for minimizing $\text{err}(h)$, we need $\text{err}(h|S)$ to be close to $\text{err}(h)$ for all the hypotheses $h$. This is given by the following guarantee.

**Theorem 13.2.** Let $\epsilon, \delta \in (0, 1)$ and consider a training set $S$ of size $m$. If $m \geq \ln(2|\mathcal{H}|/\delta)/2\epsilon^2$, then with probability at least $1 - \delta$,

$$|\text{err}(h) - \text{err}(h|S)| \leq \epsilon$$

for all $h \in \mathcal{H}$.

**Proof sketch.** We apply the additive Chernoff bound to each $h \in \mathcal{H}$, and take the union bound. □

**Example: disjunctions** Recall the concept class $\mathcal{C}$ of disjunctions over $X = \{0, 1\}^n$. Let $\mathcal{H} = \mathcal{C}$, and recall that we have a consistent learning algorithm for $\mathcal{C}$. We have $|\mathcal{H}| = |\mathcal{C}| = 3^n$ since each disjunction $f$ is defined by selecting, for each $i \in [n]$, whether $x_i$, $\bar{x}_i$, or neither appears in $f$. By ??, for $m = O((n + \log(1/\delta))/\epsilon^2)$, the algorithm produces a hypotheses $h$ with $\text{err}(h) \leq \epsilon$ with probability at least $1 - \delta$. That is, for fixed $n$, the class of disjunctions is PAC-learnable.

13.4 Occam’s razor

Occam’s razor refers the general principle that one should prefer simpler explanations. It has been echoed by many thinkers in history; for example:
• Aristotle: “Nature operates in the shortest way possible.”
• William of Occam: “Plurality is never to be posited without necessity.”
• Isaac Newton: “We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances. Therefore, to the same natural effects we must, as far as possible, assign the same causes.”
• Bertrand Russell: “Whenever possible, substitute constructions out of known entities for inferences to unknown entities.”
• Albert Einstein: “Everything should be made as simple as possible, but not simpler.”
• Richard Feynman: “The truth always turns out to be simpler than you thought.”
• The Big Aristotle: “If you don’t stick to simplicity, you’ll die a horrible death.”

For learning algorithms, one way to measure the complexity of a hypothesis class \( \mathcal{H} \) is via its descriptive bit complexity. Fix some system by which hypotheses \( h \in \mathcal{H} \) are encoded in bit strings. For a hypothesis \( h \in \mathcal{H} \), let \( |h| \) denote the number of bits needed to describe \( h \). The descriptive complexity of \( \mathcal{H} \) is defined as the maximum number of bits, \( |h| \), need to describe any hypothesis \( h \in \mathcal{H} \). The following gives generalization bounds in terms of the bit complexity of \( \mathcal{H} \).

**Corollary 13.3.** Let \( \mathcal{H} \) be a family of hypotheses of descriptive complexity \( b \). Let \( \epsilon, \delta \in (0, 1) \) and consider a training set \( S \) of size \( m \).

(i) If \( m \geq (b \ln(2) + \ln(1/\delta))/\epsilon \), then with probability at least \( 1 - \delta \), all consistent hypotheses \( h \in \mathcal{H} \) have \( \text{err}(h) \leq \epsilon \).

(ii) If \( m \geq (b \ln(2) + \ln(2/\delta))/2\epsilon^2 \), then with probability at least \( 1 - \delta \), all hypotheses \( h \in \mathcal{H} \) have \( |\text{err}(h|S) - \text{err}(h)| \leq \epsilon \).

**Proof.** We have \( |\mathcal{H}| \leq 2^b \) because \( b \) bits can describe at most \( 2^b \) hypothesis. \( \square \)

**Regularization by bit complexity.** Corollary 13.3 says that hypothesis classes of low bit complexity have better generalization error. Another way to express this theme is to say that every hypothesis \( h \in \mathcal{H} \) has generalization error proportional to its bit complexity, as follows.
Corollary 13.4. Let $\epsilon, \delta \in (0, 1)$ and consider a training set of size $m$ drawn from $D$. With probability at least $1 - \delta$, we have
\[
|err(h) - err(h \mid S)| \leq O\left(\sqrt{|h| + \ln(1/\delta)/m}\right)
\]
for all $h \in \mathcal{H}$.

Proof. For $i \in \mathbb{N}$, let $\mathcal{H}_i$ be the subset of hypothesis of bit complexity between $2^{i-1}$ and $2^i$:
\[
\mathcal{H}_i \overset{\text{def}}{=} \{h \in \mathcal{H} : 2^{i-1} < |h| \leq 2^i\}.
\]
Let $\delta_i = \delta/2^i$. By corollary 13.3, for each $\mathcal{H}_i$, we have
\[
|err(h) - err(h \mid S)| \leq O\left(\sqrt{\ln(|\mathcal{H}_i|) + \ln(\delta_i)/m}\right)
= O\left(\sqrt{\ln(2^i) + \ln(2^i/\delta)/m}\right)
= O\left(\sqrt{(|h| + \ln(1/\delta))/m}\right)
\]
for all $h \in \mathcal{H}_i$ with probability of error at most $\delta_i$. Taking the union bound over all $i$, the probability of error over all of $\mathcal{H} = \bigcup_i \mathcal{H}_i$ is $\sum_i \delta_i = \delta \sum_i 1/2^i = \delta$. □

13.5 Stronger generalization bounds via growth rate

Many learning algorithms select from a hypothesis class $\mathcal{H}$ that is geometrically fairly simple. A recurring example has been a simple consistent learner for the concept class of rectangles, which of course generalizes to higher dimensions. The perceptron algorithm looks for a hyperplane that the data. This algorithm is particularly useful when combined with techniques such as the kernel trick that first embed the data into a higher-dimensional space where the labeled data becomes linearly separable.

Recall the previous chapter that the measure of simple geometric objects is particularly well-concentrated under random sampling. Applying those techniques – namely, $\epsilon$-nets and $\epsilon$-samples – to learning gives the following generalization bounds.

To extend the notion of a growth function to a hypotheses class $\mathcal{H}$, we interpret $\mathcal{H}$ as a family of ranges corresponding to the positively labeled sets,
\[
\{h^{-1}(1) : h \in \mathcal{H}\}.
\]
 Recall that the growth function $g(n)$ of $\mathcal{H}$ is the maximum number of distinct subsets that can be induced by $\mathcal{H}$ over a set of $n$ points.
Theorem 13.5. Let $\mathcal{H}$ be a family of hypotheses with growth function $g(n)$. Let $\epsilon, \delta \in (0, 1)$. Consider a data set of size $m$.

(i) If $m \geq O\left(\ln(g(2m)/\delta)/\epsilon\right)$, then with probability at least $1 - \delta$, all consistent hypotheses $h \in \mathcal{H}$ have $\text{err}(h) \leq \epsilon$.

(ii) If $m \geq O\left(\ln(g(2m)/\delta)/\epsilon^2\right)$, then all $h \in \mathcal{H}$ have $|\text{err}(h|S) - \text{err}(h)| \leq \epsilon$.

Proof. For each hypothesis $h$, let $\tilde{h}(x) = 1 - h(x)$ denote the complementary hypothesis. Increasing $|\mathcal{H}|$ and the growth function $g(n)$ by at most a factor of 2 (why?), we assume that that $\mathcal{H}$ contains the complement $\tilde{h}$ of each hypothesis $h \in \mathcal{H}$.

Consider the (possibly infinite) point set

$$P = X \times \{0, 1\} = \{(x, y) : x \in X, y \in \{0, 1\}\}$$

We associate each hypothesis $h \in \mathcal{H}$ with the range

$$r_h = \{(x, 1 - h(x))\} \subseteq P.$$ 

The distribution $\mathcal{D}$ induces a measure $\mu_D$,

$$\mu_D(r_h) = \mathbb{P}_{(x,y) \sim \mathcal{D}} [(x, y) \in r_h]$$

The measure of a range $r_h$ equals the true error of $h$:

$$\mu_D(r_h) = \mathbb{P}_{(x,y) \sim \mathcal{D}} [(x, y) \in r_h] = \mathbb{P}_{(x,y) \sim \mathcal{D}} [h(x) \neq y] = \text{err}(h).$$

Let $R_H = \{r_h : h \in \mathcal{H}\}$. Let $f(m)$ denote the growth rate of $R_H$. The key claim is that

$$f(m) \leq g^2(n). \quad (13.2)$$

Assuming (13.2) is true, the theorem follows from the $\epsilon$-net and $\epsilon$-sample theorems from chapter 12, as we now explain.

For result (i), call a hypotheses bad if $\text{err}(h) > \epsilon$; we want to show that for a sample of the claimed size $m$, with probability at least $1 - \delta$, every bad hypotheses is inconsistent with the data. Here we observe that since $\text{err}(h) = \mu_D(r_h)$, this is equivalent to saying that the sample set is an $\epsilon$-net. The claim now follows from theorem 12.3.

For result (ii), observe that $\text{err}(h) = \mu_D(r_h)$, and for a training set $S$, $\text{err}(h|S) = \mu_S(r_h)$. Thus result (ii) is equivalent to saying that $S$ is an $\epsilon$-sample with probability at least $1 - \delta$. The sufficient bounds for $m$ now follow from theorem 12.2.
It remains to prove (13.2). Fix a set of \( m \) labeled points \( \{(x_1, y_1), \ldots, (x_m, y_m)\} \). Let \( S'_0 = \{x_i : (x_i, 1) \in S\} \) denote the set of input points labeled 0 and let \( S_0 = S_0 \times \{0\} \) denote the corresponding labeled pairs. Likewise let \( S'_1 = \{x_i : (x_i, 1) \in S\} \) and \( S_1 = S_1 \times \{1\} \) be the corresponding sets for label 1.

The total number of intersections induced by \( R_H \) on \( S, |R_H \land S| \), is bounded above by

\[
|R_H \land S| \leq |R_H \land S_0| \cdot |R_H \land S_1|,
\]

since any two intersections in \( |R_H \land S| \) must differ on either \( S_0 \) or \( S_1 \). We claim that both of the terms on the RHS are bounded above by \( g(m) \), which proves the claim.

Consider \( |R_H \land S_0| \). For any \( h \in H \), we have

\[
r_h \land S_0 = \{(x, 0) : h(x) = 1\} = (h^{-1}(1) \land S'_0) \times \{0\}.
\]

That is,

\[
|R_H \land S_0| = |H \land S'_0| \leq g(m).
\]

Similarly, for \( |R_H \land S_1| \), for each hypothesis \( h \) we have

\[
r_h \land S_1 = \{(x_i, 1) : h(x_i) = 0\} = (\bar{h}^{-1}(1) \land S'_1) \times \{1\}.
\]

Consequently

\[
|R_H \land S_1| = |H \land S'_1| \leq g(m).
\]

This completes the proof of the claim, hence the proof of the theorem.

**Example: rectangles.** Consider the concept class \( C \) of rectangles in \( \mathbb{R}^2 \). We have a consistent learning algorithm by outputting any rectangle that contains only the positively labeled points of the training set.

Rectangles have constant VC-dimension, hence polynomial growth \( g(n) = n^{O(1)} \) (cf. Sauer’s lemma, theorem 12.5). So we need \( O(\ln(1/\delta)/\epsilon) \) to guarantee that with probability at least 1 − \( \delta \), the rectangle we output has true error at most \( \epsilon \). That is, rectangles are PAC-learnable.

More generally, any concept class with fixed VC-dimension and a consistent learning algorithm is PAC-learnable.

### 13.6 Additional notes and materials

These notes are based on the first 10 lectures of [Sch19] and sections 5.1–5.7 of [BHK20]. We refer the interested reader to [Sch19] for additional background on these and other topics in learning theory.
Fall 2022 lecture materials. Click on the links below for the following files:

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

13.7 Exercises

Exercise 13.1. The definition of PAC learning requires finding a low-error hypothesis with probability at least $1 - \delta$ for arbitrary $\delta > 0$. Consider the following weaker definition of PAC learning where we drop the requirement on $\delta$: let us say that an algorithm is a weak PAC learner if for any $\epsilon > 0$, with a training set of size $\text{poly}(1/\epsilon)$, and in randomized polynomial time, it produces a hypothesis with error at most $\epsilon$ with probability at least $1/2$. Design and analyze a system that takes as input a weak PAC learning algorithm and produces a PAC learning algorithm (in the original sense).
Chapter 14

Entropy and error-correcting codes

14.1 Coding theory

Coding theory concerns the very practical problem of digital communication over imperfect lines of communication. Here we consider one particular model adopted by 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 again and eventually you start yelling. Ultimately, you are adding redundancy to try to communicate your point.

The goal of coding theory is to add enough redundancy to reliably communicate, but otherwise minimize the amount of redundancy. In particular, we have two functions, an encoder \( \mathcal{C} : \{0,1\}^m \to \{0,1\}^n \) and a decoder \( \mathcal{D} : \{0,1\}^n \to \{0,1\}^m \). The encoder takes the input message from \( x \in \{0,1\}^m \) and maps to a longer message \( \{0,1\}^n \), where \( n \geq m \). The encoded message \( \mathcal{C}(x) \) is transmitted. On the other end, the decoder receives a noise version \( y \in \{0,1\}^n \) of \( \mathcal{C}(x) \) and decodes it to some \( \mathcal{D}(y) \in \{0,1\}^m \).

To model the noisy transmission, let \( \mathcal{N} : \{0,1\}^n \to \{0,1\}^n \) be a random function that flips each bit independently with probability \( p \). We can diagram the entire
transmission process as follows.

\[
\begin{align*}
  x \xrightarrow{\text{encode}} C(x) \xrightarrow{\text{bits flip w/ prob. } p} N(C(x)) \xrightarrow{\text{decode}} D(N(C(x)))
\end{align*}
\]

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. We want high rate of transmission and low average error rate.

**Entropy** Before introducing Shannon’s theorem for codes, we have to introduce one more character: entropy.

**Definition 14.1.** 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.

We briefly 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 14.2.** Let \( n \in \mathbb{N} \) and \( p \in (0, 1) \). Then

\[
\sum_{i=0}^{pn} \binom{n}{i} \leq 2^{H(p)n}.
\]

We will prove lemma 14.2 below in section 14.2 where we discuss entropy in greater detail.
**Shannon’s theorem.** At last we state Shannon’s theorem for codes.

**Theorem 14.3 ([Sha48]).** 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 that has transmission rate at least $1 - H(p) - \delta$ and average error rate at most $\delta$.

2. For all fixed $\delta > 0$, and $n$ sufficiently large any coding scheme with transmission rate at least $1 - H(p) + \delta$ has average error rate greater than $\delta$.

### 14.1.1 Shannon’s Upper Bound

In this section, we prove Shannon’s upper bound - the first claim in theorem 14.3. We first state the part that is relevant.

**Theorem 14.4.** 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 $m = (1 - H(p) - \delta)n$. Rather than propose a specific code, we define $C : \{0, 1\}^m \to \{0, 1\}^n$ to 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 $x' \in \{0, 1\}^m, x' \neq x$ has its codeword $C(x')$ within $(1 + \epsilon)pn$ bits of $N(C(x))$.

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 \( \mathcal{C} : \{0, 1\}^m \rightarrow \{0, 1\}^n \), and the noise \( \mathcal{N}(\mathcal{C}(x)) \), we have
\[
P_{x, \mathcal{C}, \mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x] \leq \delta. \tag{14.1}
\]
Inequality (14.1) above seems similar to the low average error guarantee we seek. Subtly, however, the claim we want to prove is that there exists a fixed (and no longer random) code \( \mathcal{C} \) with low average error, where as the LHS (14.1) is also randomized over all possible codes \( \mathcal{C} \). To isolate a specific code \( \mathcal{C} \) with average error \( \delta \), we rewrite (14.1) as
\[
E_{\mathcal{C}}[\text{average error of } \mathcal{C}] = E_{\mathcal{C}}[P_{x, \mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x]] \leq \delta.
\]
To dramatically complete the proof: by the probabilistic method, there exists an encoding \( \mathcal{C} : \{0, 1\}^m \rightarrow \{0, 1\}^n \) such that the average error is \( \leq \delta \).

Claim 1. For sufficiently large \( n \), with probability of error \( \leq \delta/2 \), no other point \( x' \in \{0, 1\}^m \), \( x' \neq x \) is within \( (1 + \epsilon)pn \) bits of \( \mathcal{N}(\mathcal{C}(x)) \).

We prove the claim conditional on \( y = \mathcal{N}(\mathcal{C}(x)) \); the unconditional claim immediately follows. Fix \( y = \mathcal{N}(\mathcal{C}(x)) \). Consider any other input point \( x' \in \{0, 1\}^m \). By construction, \( \mathcal{C}(x') \) is selected uniformly at random from \( \{0, 1\}^n \). Therefore
\[
P[\|\mathcal{C}(x') - \mathcal{C}(x)\|_0 \leq (1 + \epsilon)pn] = 2^{-n} \sum_{i=0}^{(1+\epsilon)pn} \binom{n}{i} \leq 2^{(H((1+\epsilon)p)-1)n}.
\]
By the union bound, we have
\[
P[\|\mathcal{C}(x') - \mathcal{C}(x)\|_0 \leq (1 + \epsilon)pn \text{ for some } x' \neq x] \leq 2^{m + H((1+\epsilon)p)-1)n}.
\]
The RHS is \( \leq \delta/2 \) iff
\[
m \leq (1 - H((1 + \epsilon)p)n - 1 - \log(1/\delta),
\]
which occurs iff
\[
\text{(transmission rate)} = \frac{m}{n} \leq 1 - H((1 + \epsilon)p) - O\left(\frac{\log(1/\epsilon)}{n}\right).
\]
By choice of \( \epsilon \), 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 \( \leq \frac{\delta}{2} \), a noisy transmission \( N(y) \) flips at most \((1 + \epsilon)pn \) bits in \( y \).

We have
\[
\lim_{n \to \infty} P[(\# \text{ bits flipped}) \geq (1 + \epsilon)pn] \leq \lim_{n \to \infty} e^{-\epsilon^2pn/2} = 0.
\]
where (a) applies the Chernoff inequality.

14.1.2 Lower bounds for Shannon’s capacity

**Theorem 14.5.** 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

\[
1 - H(p) + \delta.
\]

**Proof.** Let \((C : \{0, 1\}^m \to \{0, 1\}^n, D : \{0, 1\}^n \to \{0, 1\}^m)\) 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 \( \bar{E} \) be the event that \( N(C(x)) \) differs in at least \((1 - \epsilon)pn \) bits from \( C(x) \) and no more than \((1 + \epsilon)pn \) bits from \( C(x) \). Let \( \bar{E} \) be the complementary event.

We have
\[
(\text{average correctness rate}) \leq P[\bar{E}] + P_{x,C}[D(N(C(x))) = x | E].
\]

For the first term, we have
\[
P[\bar{E}] = P[\|N(C(x)) - C(x)\|_0 < (1 - \epsilon)pn] + P[\|N(C(x)) - C(x)\|_0 > (1 + \epsilon)pn]
\leq 2e^{-\epsilon^2pn/3}
\]
by Chernoff bounds. In particular, for fixed \( \epsilon > 0 \),
\[
\lim_{n \to \infty} P[\bar{E}] = 0.
\]

To bound the second term, for each \( x \), let
\[
Y_x = \{y \in D^{-1}(x) : (1 - \epsilon)pn \leq \|x - y\|_0 \leq (1 + \epsilon)pn\}.
\]

For fixed \( x \), under event \( E \), \( Y_x \) represents all the possible points for \( N(C(x)) \) that would be decoded to \( x \). We have
\[
P_{x,N}[D(N(C(x))) = x | E] = \frac{1}{2^m} \sum_{x \in \{0, 1\}^m} \sum_{y \in Y_x} P[N(C(x)) = y].
\]

217
For each $x$, and for each $y \in Y_x$,

$$P_{N}[N(C(x)) = y] \leq p^{(1-\epsilon)pn}(1 - p)^{(1-(1-\epsilon)p)n} = 2^{-H(p)n}\left(\frac{1-p}{p}\right)^{\epsilon pn}$$

Here (a) is because $y$ differs in at least $(1 - \epsilon)pn$ bits. We now have

$$P_{x,N}[D(N(C(x))) = x \mid E] \leq \frac{1}{2^m} \sum_{x \in \{0,1\}^m} e^{-H(p)n}\left(\frac{1-p}{p}\right)^{\epsilon pn} |Y_x|$$

$$\leq 2^{-(1-H(p))n-m}\left(\frac{1-p}{p}\right)^{\epsilon pn}$$

$$\leq 2^{-\delta n}(1-p)^{\epsilon p n} = 2^{(\epsilon \log(\frac{1-p}{p})-\delta)n}.$$ 

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 $n \to \infty$.

\[\square\]

### 14.2 Entropy

In this section, we explore the some of the many interesting properties of entropy. We restate the definition for the reader’s convenience.

**Definition 14.1.** 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\(^1\) the *shock* of \( X \), \( S_X > 0 \), as the random variable that, if \( X = x \), takes the value
\[
S_X = \frac{1}{\mathbb{P}[X = x]}.
\]
Here \( \mathbb{P}[X = x] \) refers to the *a priori* probability of \( X \) equaling \( x \). Then the entropy of \( X \) is
\[
H(X) = \mathbb{E}[\log(S_X)].
\]

### 14.2.1 Concavity and the maximality principle.

Recall that a function \( f : [a, b] \to \mathbb{R} \) is *concave* if for all \( x, y \in [a, b] \) and \( p \in [0, 1] \), we have
\[
rf(x) + (1 - r)f(y) \leq f(px + (1 - p)y).
\]
By induction, we can extend this to finite convex combinations of points. Let \( x_1, \ldots, x_n \in [a, b] \) and \( p_1, \ldots, p_n \geq 0 \) with \( p_1 + \cdots + p_n = 1 \). Then we have
\[
p_1f(x_1) + \cdots + p_nf(x_n) \leq f(p_1x_1 + \cdots + p_nx_n). \quad (14.2)
\]
Let \( f : [a, b] \to \mathbb{R} \) be a function and let \( X \in [a, b] \) be a discrete random variable taking on a finite number of values. Say \( X \) takes on \( n \) values \( x_1, \ldots, x_n \) with probabilities \( p_1, \ldots, p_n \) respectively. Then \((14.2)\) is the same as saying that
\[
\mathbb{E}[f(X)] \leq f(\mathbb{E}[X]).
\]
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 14.6.** Let \( X \in [a, b] \) be a random variable, and let \( f : [a, b] \to \mathbb{R} \) be concave. Then
\[
\mathbb{E}[f(X)] \leq f(\mathbb{E}[X]).
\]

**Proof.** Suppose \( X \) takes on only two values, \( a \) and \( b \), with probability \( p \) and \( (1 - p) \) respectively. Then
\[
\mathbb{E}[f(X)] = pf(a) + (1 - p)f(b) = f(pa + (1 - p)b) = f(\mathbb{E}[X]).
\]
We can extend the argument to any finite number of values by induction. \( \square \)

---

\(^1\)This is not a standard terminology.
Lemma 14.7. Over all discrete distributions over $n$ values, entropy is maximized by the uniform distribution, which has entropy is $\log(n)$.

Proof. Suppose $X$ takes on at most $n$ different values $x$. Then

$$H(X) = \mathbb{E}[\log(S_X)] \overset{(a)}{\leq} \log(\mathbb{E}[S_X]) \overset{(b)}{=} \log(|X|)$$

(a) is by Jensen’s inequality. (b) is because

$$\mathbb{E}[S_X] = \sum_x \mathbb{P}[X = x] = n.$$

On the other hand, if $X$ is the uniform distribution over $n$ values $x_1, \ldots, x_n$, then

$$H(x) = \sum_i \mathbb{P}[X = x_i] \log\left(\frac{1}{X = x_i}\right) = \sum_i \frac{1}{n} \log(n) = \log(n).$$

$\square$

14.2.2 Conditional entropy

Let $(X, Y)$ be jointly distributed random variables. Conditional on $X$, $Y$ is a random variable with a well defined entropy $H(Y)$ (given $X$).

Definition 14.8. The conditional entropy of $Y$ on $X$ is defined as

$$H(Y \mid X) = \mathbb{E}_X[H(Y) \mid X]$$

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

$$S_{Y \mid X} = \frac{1}{\mathbb{P}[Y = y \mid X = x]}$$

where $\mathbb{P}[Y = y \mid X = x]$ is the a priori probability given only $X = x$. Then

$$H(Y \mid X) = \mathbb{E}_X[\mathbb{E}[\log(S_{X \mid Y})]] = \mathbb{E}_{X,Y}[\log(S_{X \mid Y})]$$

We have the following identity that breaks the joint entropy into two entropy terms.

Lemma 14.9. $H(X, Y) = H(Y \mid X) + H(X)$
14. Entropy and error-correcting codes

14.3 Principle of Independence

Proof. Observe that conditional on $X = x$ and $Y = y$, we have

$$S_{X,Y} = \frac{1}{P[X = x, Y = y]} = \frac{1}{P[X = x] P[Y = y | X = x]} = S_X S_{Y|X}.$$ 

Thus

$$H(X, Y) = E_{X,Y} [\log (S_{X,Y})] = E_{X,Y} [\log (S_{Y|X}) + \log (S_X)] = H(Y | X) + H(X),$$

as desired.

### Lemma 14.10

Let $(X, Y)$ be jointly distributed. Then $H(Y | X) \leq H(Y)$

Proof. We have

$$H(Y | X) = E_{X,Y} [\log (S_{Y|X})] = E_Y \left[ E_X [\log (S_{Y|X}) | Y] \right] \leq (a) \leq E_Y \left[ \log \left( E_X [S_{Y|X} | Y] \right) \right] \leq H(Y).$$

(a) is by Jensen’s inequality. (b) is because, conditional on $Y = y$, we have

$$E_X [S_{Y|X} | Y] = \sum_x P[X = x | Y = y] P[Y = y | X = x] \overset{(c)}{=} \sum_x \frac{p(x)}{p(y)} = 1.$$

(c) substitutes Bayes’ law:

$$P[X = x | Y = y] P[Y = y] = P[X = x, Y = y] = P[Y = y | X = x] P[X = x].$$

Combining the principle of independence with lemma 14.9 gives the following principle called maximality of independence.

### Lemma 14.11

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.

Maximality of independence is also captured by following inequality, also called the subadditivity property of entropy.

### Lemma 14.12

Let $(X, Y)$ be a joint distribution of discrete random variables. Then

$$H(X, Y) \leq H(X) + H(Y).$$
14.3.1 Bounding sums of binomial coefficients

Finally, let us prove lemma 14.2 which we recall was the key to the proof of Shannon’s upper bound.

**Lemma 14.2.** Let \( n \in \mathbb{N} \) and \( p \in (0, 1) \). Then

\[
\sum_{i=0}^{np} \binom{n}{i} \leq 2^{H(p)n}.
\]

**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. \( \square \)

14.3.2 What is entropy, really?

We close with a quote from Shannon [TM71].

“...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”.

14.4 Additional notes and materials

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

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

**Exercise 14.1.** In section 14.1, we discussed coding schemes for achieving low average error rates. Recall that an average error rate of \( \delta \) means that the transmission failure probability is averaged over all \( x \in \{0,1\}^m \):

\[
\text{average error} = \mathbb{E}_{x} [\mathbb{P}_{\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x]] = \mathbb{P}_{x,\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x].
\]

By contrast, a uniform error of \( \delta \) means that for every \( x \in \{0,1\}^m \), the transmission failure probability is at most \( \delta \): that is,

\[
\text{uniform error} \overset{\text{def}}{=} \max_{x} \mathbb{P}_{\mathcal{N}}[\mathcal{D}(\mathcal{N}(\mathcal{C}(x))) \neq x] \leq \delta.
\]

Prove Shannon’s upper bound (theorem 14.3) for uniform error instead of average error. That is, show that for all fixed \( \delta > 0 \), there exists a coding scheme \( (\mathcal{C} : \{0,1\}^m \rightarrow \{0,1\}^n, \mathcal{D} : \{0,1\}^n \rightarrow \{0,1\}^m) \) (for \( m,n \) sufficiently large) with uniform error \( \delta \) and transmission rate at least \( 1 - H(p) - \delta. \)

**Exercise 14.2.** In section 14.1, we develop redundant codes that are extremely efficient w/r/t their transmission rate. Another problem, moving in sort of the opposite direction, is compression.

Here we consider compression in the following model. Let \( \Sigma \) 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

\[
\mathcal{C} : \Sigma \rightarrow \{0,1\}^*
\]

assigning bit strings (of varying length) to letters such that no code \( \mathcal{C}(x) \) \( (x \in \Sigma) \) is a prefix to another code \( \mathcal{C}(y) \) \( (y \in \Sigma) \). Prefix codes are particularly easy to decode. As we scan the bits of an encoded message, as soon as we see a string that matches the code of a letter, we immediately decode that the scanned bits to the letter. We then continue to scan the rest of the bits as the beginning of the code of a new letter.

For example, the most straightforward prefix code would be to assign each letter in \( \Sigma \) a different bit string with \( \lceil \log n \rceil \) bits.

---

2 Hint: Given a code with average error rate \( \delta \), how many of the input words \( x \) have transmission failure probability greater than \( 2\delta \)?
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 $1/2^{i_x}$ 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.
{A,B,C,D,E,F,G}

0   1

{A,B}    {C,D,E,F,G}

0   1

A     B
1/4   1/4

{E,F,G}    {C,D}

0   1

E    F
1/8   1/16

1/8

G
1/16
Chapter 15

Lovász local lemma and resampling

15.1 Resampling $k$-SAT

Recall that in the $k$-SAT problem, we are given a CNF formula $f(x_1, \ldots, x_n)$ with $m$ clauses $C_1, \ldots, C_m$ and $k$ (distinct) variables per clause. In max $k$-SAT, the goal is to satisfy as many clauses as possible, and previously we showed that a random assignment satisfies $(1 - 2^{-k})m$ clauses in expectation. More precisely we showed that each clause is satisfied with probability $1 - 2^{-k}$.

Now suppose $m < 2^k$. By the union bound, we have

$$\Pr[f(x_1, \ldots, x_n) = \text{false}] \leq \sum_{i=1}^{m} \Pr[\text{ith clause is unsatisfied}] = \frac{m}{2^k} < 1.$$ 

Therefore, $f$ must be satisfiable!

Of course, when $m > 2^k$, a simple union bound does not show that $f$ is satisfiable. In this chapter, we explore a probabilistic technique called the local lemma, that offers an alternative criteria for proving that a $k$-SAT formula is satisfiable, independent of $m$.

The theorem we unveil below is constructive and comes with a simple randomized algorithm for $k$-SAT which we now describe. The algorithm is called the (randomized) resampling algorithm and was described by [Mos09]. Given a $k$-SAT formula $f(x_1, \ldots, x_n)$, we first assign $x_1, \ldots, x_n \in \{\text{true, false}\}$ independently and uniformly at random. Then, as long as there is a clause $C$ that is unsatisfied, we resample all the variables in $C$ independently and uniformly at random. The algorithm continues to resample unsatisfied clauses until all clauses are unsatisfied. (A priori it seems that the algorithm may never terminate.)

Surprisingly, under certain conditions independent of $m$ and $n$, and which are easy to verify, $f$ always has a satisfying assignment, and the resampling algorithm described above terminates in polynomial time in expectation. Below, we say that two
clauses $C_i$ and $C_j$ of $f$ intersect if there is a variable $x_k$ appearing (as is, or negated) in both $C_i$ and $C_j$.

**Theorem 15.1.** Let $f(x_1, \ldots, x_n)$ be a $k$-SAT CNF with $m$ clauses, such that each clause intersects at most $d$ other clauses. If $d \leq 2^k/e - 1$, then $f$ is satisfiable. Moreover, a satisfying assignment can be computed by the resampling algorithm while sampling at most $m/d$ clauses in expectation.

The most striking feature of theorem 15.1 is that the criteria for satisfiability — $d \leq 2^k/e - 1$ where $d$ bounds the number of clauses overlapping with a single clause — is local. It is easy to inspect a CNF formula and compute $d$ and see if theorem 15.1 applies.

### 15.2 The Lovász Local Lemma

The $k$-SAT result described above is just one example of a more general technique called the *Lovász local lemma (LLL)* [EL75]. It applies in a more abstract setting where we have a family of events we want to avoid, and assuming dependency among the events is limited in a particular way.

Formally, let $A_1, \ldots, A_n$ denote events in an arbitrary probability space. We write $A_i \sim A_j$ when $A_i$ and $A_j$ are not mutually independent, and $A_i \not\sim A_j$ otherwise. The question is whether it is possible for none of the events $A_1, \ldots, A_n$ to occur simultaneously; i.e., if $P[\overline{A_1}, \ldots, \overline{A_n}] > 0$. [EL75] gave the following sufficient condition which depends only on the local dependencies of each event $A_i$ to other events $A_j$.

**Theorem 15.2 ([EL75]).** Suppose there exists values $x(A_1), \ldots, x(A_n) \in (0, 1)$ such that for all events $A_i$,

$$P[A_i] \leq x(A_i) \prod_{A_j \sim A_i, A_j \neq A_i} (1 - x(A_j)).$$ (15.1)

Then $P[\overline{A_1}, \ldots, \overline{A_n}] > 0$.

A simpler ("symmetric") version of theorem 15.2 that applies to the $k$-SAT problem is as follows.

**Corollary 15.3.** Suppose that each event $A_i$ is mutually independent of all but at most $d$ other events $A_j$, and $P[A_i] \leq p$ for all events $A_i$. If $ep(d + 1) \leq 1$, then $P[\overline{A_1}, \ldots, \overline{A_n}] > 0$.

---

1Erdős attributes the technique to Lovász and helped popularize the name. LLL is also referred to as just the *local lemma* in the literature.
Proof. The claim follows from theorem 15.2 for $x(A_i) = 1/(d + 1)$.

This establishes the fact that any $k$-SAT formula with at most $2^k/e$ dependencies per clause is satisfiable. Theorem 15.2 has many other applications both in computer science and combinatorics, such as in proving extremal graph coloring bounds (e.g., [AS16; MR02]). For $k$-SAT, however, it does not provide an algorithm for actually computing such an assignment.

Developing an algorithmic LLL was an important open question. Progress on this question starts in 1991 when [Bec91] analyzed an algorithm for a hypergraph 2-coloring problem that can also be addressed by LLL. Subsequent developments followed in [Alo91; CS00; MR98; Sri08].

In 2009, a breakthrough by Moser [Mos09] proved theorem 15.1 up to a constant factor in the bound for $d$. This work proposed the simple resampling algorithm described above and also a surprising proof technique now called the “entropy compression argument”. This was followed up by [MT10] which extended the techniques to the more general setting of theorem 15.2 (via a related but different proof), and obtained tight bounds.

To present the constructive version of theorem 15.2 we assume a more concrete setting where the events are driven by an underlying set of independent random variables. Let $\mathcal{V}$ be a finite collection of mutually independent random variables in a fixed probability space. Let $A_1, \ldots, A_n$ be a set of events where each event $A_i$ is determined by a subset of variables $S_i \subseteq \mathcal{V}$. (For $k$-SAT, $\mathcal{V}$ corresponds to the boolean variables $x_1, \ldots, x_n$, and each clause $C_i$ is associated with the event $A_i$ where $C_i$ is not satisfied.) The concrete goal is to compute a configuration of $\mathcal{V}$ so that none of the events $A_1, \ldots, A_n$ occur.

The resampling algorithm extends to this more general setting as follows. We initially sample each variable $X \in \mathcal{V}$ independently from their respective distributions. While there is an event $A_i$ induced by the current sample of variables, we resample the variables in $S_i$. The algorithm terminates when none of the events $A_1, \ldots, A_n$ occur.

The following theorem from [MT10] states that the resampling algorithm gives an algorithmic LLL.

**Theorem 15.4.** Given the setting of theorem 15.2, the resampling algorithm computes an assignment of values to $\mathcal{V}$ inducing $\bar{A}_1, \ldots, \bar{A}_n$, while resampling each event $A_i$ at most $x(A_i)/(1 - x(A_i))$ times in expectation.

The rest of this chapter is focused on proving theorems 15.2 and 15.4 in the more concrete setting with variables $\mathcal{V}$. We follow the proof of [MT10]. We mention that
theorem 15.2 can be proven without the existence of $V$ and we refer to [AS16, §5.1] for this proof (which is shorter).

15.3 Analysis of the resampling algorithm

The high-level goal is to bound the total number of resampling steps taken by the resampling algorithm, in expectation.

Recall that the algorithm selects events to sample in arbitrary order. For the sake of analysis we fix any deterministic or randomized mechanism to select these events. Each instance of the resampling algorithm can be associated with a log where we list the sampled events in chronological order. Note that a single event may occur multiple times in the log. Let $L \in \mathbb{Z}_{\geq 0} \cup \{\infty\}$ denote the (randomized) length of the log. We want to bound the expected length of the log, $E[L]$.

**Witness trees.** Moser and Tardos’s [MT10] argument is based on analyzing auxiliary rooted trees called *witness trees*. A *witness tree* is defined as a (nonempty) rooted tree where each node is labeled by an event $A_i$. We only consider witness trees with the following additional properties.

(a) If a node labeled $A_i$ is a child of a node labeled $A_j$, then $A_i \sim A_j$.

(b) If two nodes labeled $A_i$ and $A_j$ have the same depth, then $A_i \nsim A_j$.

For each prefix of the log, containing (say) $i$ entries, we associate a witness tree $T_i$ constructed as follows. Let $A_{i_1}, \ldots, A_{i_i}$ be the event labels of the first $i$ entries in the log, where $i \leq L$. We process the events in reverse order. First, we create a root labeled by $A_{i_i}$. Then, for $j = i - 1$ down to 1, if $A_{i_j}$ shares variables with an event labeling of a node $N$ in the tree, then $N$ be the node of greatest depth, and create a new child of $N$ labeled by $A_{i_j}$. (If $A_{i_j}$ is independent of all nodes in the tree, then we do not create a node for $A_{i_j}$.)

This process produces a rooted tree with at most $i$ nodes. We see that the tree satisfies property (a) above because a node with label $A_i$ is made a child of a node with label $A_j$ only if $A_i$ and $A_j$ share at least one variable. Property (b) follows from the fact that when introducing a node $n$ with $A_i$, we always make $n$ a child of the node $n'$ with maximum depth among those labeled by an event $A_j$ that share a variable with $A_i$.

We highlight one observation about this construction that will be particularly useful later on.

**Observation 15.5.** Consider a prefix of the log with witness tree $T$. Let $e_1$ and $e_2$ be two entries in the log with $e_1$ listed before $e_2$, labeled by events $A_{i_1}$ and $A_{i_2}$, respectively.
If $A_{i_1} \sim A_{i_2}$, and $e_2$ is a node in the witness tree $T$, then $e_1$ is also a node in the witness tree at depth strictly greater than $e_1$.

For $i = 1, 2, \ldots$, when $i \leq L$, let $T_i$ be the tree certificate associated by the first $i$ events $A_1, \ldots, A_i$ in the log. For indices $i > L$, we let $T_i$ denote a null value $\emptyset$ (not equal to any witness tree). We can express the expected length of the log in terms of the probabilities of each tree arising at each prefix of the log, as follows:

$$E[L] = \sum_{i \in \mathbb{N}} \sum_T P[T_i = T], \quad (15.2)$$

where the inner sum is over all certificate trees.

**Lemma 15.6.** For all distinct $i, j \in [L]$, $T_i \neq T_j$.

**Proof.** Let $i < j$. To have $T_i = T_j$, the $i$th and $j$th entry of the log must correspond to the same event $A_k$.

Suppose that is the case. Then every node added to $T_i$, including the node corresponding to the $i$th entry, will also be added to $T_j$. Since $T_j$ also includes a node for the $j$th entry, this shows that $|T_j| > |T_i|$, hence $T_j \neq T_i$.

The fact the all $T_i$’s are distinct allows us to simplify (15.2) to

$$E[L] \leq \sum_T P[T = T_i \text{ for some } i].$$

Next we analyze the probability that a fixed certificate tree $T$ arises (anywhere) in the log. For a tree certificate $T$, let

$$p(T) = \prod_{\text{labels } A_i \text{ in } T} P[A_i],$$

where the product ranges over all events $A_i$ labeling nodes in $T$, with multiplicity.

**Lemma 15.7.** $P[T_i = T \text{ for some } i] \leq p(T)$.

**Proof.** Consider the following random procedure which we call checking $T$. Let $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ list the labels of $T$ in decreasing order of depth (starting with a leaf, and ending at the root). For each $A_{i_j}$ in order, we resample $S_{i_j}$. We say that this procedure of checking $T$ passes if for each $A_{i_j}$, resampling $S_{i_j}$ induces $A_{i_j}$.

The probability that $T$ passes the check is exactly $p(T)$. We want to show that the probability of $T$ occurring in the log is bounded above by the probability that $T$ passes the check.
Recall that the algorithm resamples the variables \( V \) repeatedly, and each variable is sampled independently. We imagine generating, for each variable, an entire (infinite) sequence of samples ahead of time, and then running the resampling algorithm on these fixed sequences of samples. Here the first sample of each variable is used for the initialization. In this scheme, we can associate to each log-entry \( e \), corresponding to resampling for an event \( A_i \), the set of values for \( S_i \) before resampling that induced \( A_i \). Observe that these corresponding sets of values for the log entries are disjoint subsets of the fixed set of samples, and each set of values induces the event corresponding to that entry.

Now fix a set of samples which induces a particular log. Whenever \( T \) occurs on the log, we check \( T \) using the same fixed set of samples. That is, when the check processes \( A_{ij} \) and resamples \( S_{ij} \), for each variable in \( S_{ij} \), we use the next sample for that variable not used from processing other events previously in the check. We will show that \( T \) occurs in the log only if \( T \) passes the check (with respect to the fixed sample). This implies that the probability that \( T \) occurs in the log is bounded above by the probability that \( T \) passes the check, and completes the proof.

Fix a node \( N \) in the tree \( T \) which corresponds to a fixed entry \( e \) in the log. Let \( A_i \) be the label of \( N \) and \( e \), and let \( X \in S_i \) be any variable in its support. We want to show that both the value for \( X \) associated with the event \( e \), and the value of \( X \) drawn by the checking procedure when processing \( n \), corresponds to the same sample of \( X \). Now, the resampling procedure uses the \( j \)th sample of \( X \) where \( j \) is the number of entries in the log up to and including \( A_i \) labeled by an event that depends on \( X \). The tree check uses the \( k \)th sample of \( X \) where \( k \) is the number of events chronologically before and including \( A_i \) in the checking process that has \( X \) in its support. If \( j = k \), then since we know that the sample corresponding to \( e \) induces \( A_i \), we know that the checking procedure will also induce \( A_i \) and processing the node \( n \). Taken over all nodes \( n \), we conclude that the \( T \) passes the check.

To show that \( j = k \), consider any log entry \( e' \) that (a) comes before \( e \) and (b) is labeled by an event \( A_j \) that also depends on \( X \). As observed above, the construction of the tree certificate implies that there is also a node \( N' \) corresponding to \( e' \), and \( N' \) has depth strictly greater than \( N \). Meanwhile any entry \( e' \) that (a) comes after \( e \) and (b) is labeled by an event depending on \( X \) either not in the tree or appearing at a depth strictly less than \( n \). This implies that \( j = k \) in the sense described above, and completes the proof.

It remains to bound the sum of \( p(T) \) over all witness trees \( T \). We break this down by the event labeling the root.
Lemma 15.8. Let $\mathcal{T}_i$ be the family of witness trees with root labeled by $A_i$. Then

$$\sum_{T \in \mathcal{T}_i} p(T) \leq \frac{x(A_i)}{1 - x(A_i)}.$$

Proof. Consider the following Galton-Watson branching process producing a tree in $\mathcal{T}_i$.

In the first round we introduce a root labeled $A_i$. Each round, for each node $N$ introduced in the previous round labeled by an event $A_j$, we do the following. For each event $A_k$ with $A_k \sim A_j$ (including $A_k = A_j$), with probability $x(A_k)$, we introduce a child of $N$ labeled by $A_k$.

The process terminates when no new vertices are introduced in a single round. (The process may not necessarily terminate).

Fix a particular tree $T \in \mathcal{T}_i$. Let $q_T$ denote the probability that $T$ is produced by the Galton-Watson process. For each node $n \in T$, let $A_n$ denote the event labeling $n$. Let $B_n$ denote the set of events $A_k$ such that $A_k \sim A_n$, but the process did not produce a child of $n$ with label $A_k$. We have

$$q_T = \frac{1}{x(A_i)} \prod_{n \in T} x(A_n) \prod_{A_k \in B_n} (1 - x(A_k)).$$

Now, we have

$$\prod_{n \in T} \prod_{A_k \in B_n} (1 - x(A_k)) = (1 - x(A_i)) \frac{\prod_{n \in T} \prod_{A_k: A_k \sim A_n} (1 - x(A_k))}{\prod_{n \in T} (1 - x(A_k))}$$

since every $A_k$ omitted from $B_n$ appears as the label of another distinct node, and these omitted events cover all the nodes except the root. This leads to the cleaner bound,

$$q_T = \frac{1 - x(A_i)}{x(A_i)} \prod_{n \in T} x(A_n) \prod_{A_k \sim A_n} (1 - x(A_k))$$

$$= \frac{1 - x(A_i)}{x(A_i)} \prod_{n \in T} x(A_n) \prod_{A_k \sim A_n: A_k \neq A_n} (1 - x(A_k)).$$

Now, by plugging in the inequality (15.1) assumed in the theorem, we have

$$p(T) = \prod_{\text{node } n \in T} P[A_n] \leq \prod_{n \in T} x(A_n) \prod_{A_j \sim A_n: A_j \neq A_n} (1 - x(A_j)) = \frac{x(A_i)}{1 - x(A_i)} q_T.$$
Summing over all $T \in \mathcal{T}_i$, we have
\[ \sum_{T \in \mathcal{T}_i} p(T) \leq \frac{x(A_i)}{1 - x(A_i)} \sum_{T \in \mathcal{T}_i} q_T \leq \frac{x(A_i)}{1 - x(A_i)}, \]
where the last inequality observes that the Galton-Watson produces at most one tree, hence all probabilities $q_T$ sum to at most 1.

Now we complete the proof of theorem 15.2. For the expected length of the log, we now have
\[ \mathbb{E}[L] \leq \sum_T P[T_i = T \text{ for some } i] \leq \sum_T p(T) \leq \sum_{i=1}^{n} \sum_{T \in \mathcal{T}_i} p(T) \leq n \sum_{i=1}^{n} \frac{x(A_i)}{1 - x(A_i)}. \]
To more precisely bound the expected number of times we resample an event $A_i$, we observe that $A_i$ is resampled only if a tree $T \in \mathcal{T}_i$, with root labeled by $A_i$, is produced by the log. Thus
\[ \mathbb{E} \left[ \# \text{ times } A_i \text{ is resampled} \right] \leq \sum_{T \in \mathcal{T}_i} P[T_i = T \text{ for some } i] \leq \sum_{T \in \mathcal{T}_i} p(T) \leq \sum_{i=1}^{n} \frac{x(A_i)}{1 - x(A_i)}, \]
as desired.

15.4 Additional notes and materials

We refer the reader to [MT10] for extensions including parallelization, a lopsided LLL condition, and derandomization. Additional applications of the local lemma can be found in [AS16]. We refer to [Tao09] for an alternative, information-theoretic perspective on Moser’s [Mos09] original proof technique.

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

15.5 Exercises

**Exercise 15.1.** A hypergraph is a generalization of an undirected graph where each edge may have more than 2 endpoints. A hypergraph has uniform rank $k$ if every edge contains exactly $k$ points. A hypergraph is $k$-regular if each vertex is incident to exactly $k$ edges.
Design and analyze an algorithm that, for \( k \) sufficiently large\(^2\), takes as input a \( k \)-regular hypergraph of uniform rank \( k \) and either: (a) outputs a vertex coloring such of 2 colors so that no edge is monochromatic, or (b) declares that no such coloring exists.

**Exercise 15.2.** Recall that a *proper vertex coloring* of an undirected graph \( G \) is one where no two adjacent vertices have the same coloring. Here we consider *vertex list-coloring*, where each vertex \( v \) is given a finite list \( L_v \) of colors, and we want a proper vertex coloring where each vertex is assigned a color from a list.

Design and analyze an algorithm that, given an undirected graph \( G = (V, E) \) with maximum degree \( \Delta \), and lists of colors \( L_v \) for each vertex \( v \) of size \( |L_v| \geq 10\Delta \), either computes a proper vertex list-coloring or declares that no such coloring exists.

\(^2\)That is, the algorithm should work for all \( k \geq c \) for some universal constant \( c \). \( c = 9 \) is possible.
Figure 16.1: Map of a small part of the world wide web around wikipedia.org [CW].
16.1 Ranking the web

The world wide web is a large and messy place. As of March 2022, http://worldwidewebsize.com estimates that there are 2.97 billion indexed webpages. Even more amazing is that modern search engines, starting with Google, are able to process, organize and index this nearly unbounded corpus and make it useful. You can query for a topic of interest, and the search engine returns a long list of relevant websites, almost immediately. More often than not, you find what you are looking for within the first few listed results. This is utterly amazing, and we basically take it for granted.

It is one thing to identify all the web pages containing (or relevant to) a search query. This requires crawling the internet, and building a huge index that roughly identifies which keywords appear where. Some of the randomized data structures discussed earlier may be helpful for managing this task. But even within a search query, there seems to be an unlimited number of pages about a given topic, and a lot of it is junk. There is still another challenge to identify the best pages for the query. How do we separate the good websites from the bad? We should keep in mind the scale of the world wide web. It is pointless to try to evaluate the websites individually. This is a large scale ranking problem.

Modern search engines are based on the idea that the link structure of the world wide web reveals some sense of importance among the websites. When we write a paper, we cite the references that support or inform our argument. Likewise, web sites link to other websites and thereby implicitly bestow some approval. Another appeal of analyzing links is that we can model everything in basic graph theory, where we have good algorithms and sound analysis. The link structure gives a good starting point for our first idea for ranking webpages.

**Idea 1. Score each website equal to the number of other webpages linking to it.**

\[
score_1(v) = \sum_{(u,v) \in E} 1.\]

One feature of score1 is that every link out of a vertex u is worth 1 point. But if u has many outgoing links, shouldn’t that dilute the “approval” bestowed by u? As an analogy, suppose we have two lists of movies. One lists the top 10 movies of all time, and the other lists the top 100 movies of all time. Shouldn’t it be worth more to be on the first list? Our next proposal for ranking scales down the value of a link \((u,v)\) by the number of outgoing edges of u, so that the total sum of links leaving u is 1.

Let \(d^+(u)\) denote the number of edges leaving a vertex u, a.k.a. the out-degree.
Idea 2. Score each website $v$ as the sum, over all other webpages $u$ linking to $v$, of $1$ divided by the number of outgoing links from $u$:

$$\text{score}_2(v) = \sum_{w:(u,v) \in E} \frac{1}{d^+(u)}.$$ 

Unfortunately, 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 score$_2(v)$. This suggests that, when evaluating a link $(u,v)$, we should consider whether $u$ is much of an authority to begin with.

Idea 3. Score each website $v$ equal to the weighted sum, over all other webpages $u$ linking to it, of the score of $u$ divided by the number of outgoing links from $u$:

$$\text{score}_3(v) = \sum_{w:(u,v) \in E} \frac{\text{score}_3(u)}{d^+(u)}.$$ 

score$_3(v)$ attain a sort of self-consistent nirvana. For example, the weight of a link $(u,v)$, in attributing authority to $v$, is adjusted in proportion to the authority of $u$. The total authority distributed by a webpage $u$ is exactly equal to $u$’s own authority, score$_3(u)$.

While this recursive relationship is appealing, there is no reason, a priori, why such scores should exist.

Today’s discussion is about how $\{\text{score}_3(v), v \in V\}$ does exist, and the structure and interpretations thereof. This happy miracle is entirely due to the fact that the values $\{\text{score}_3(v), v \in V\}$ satisfy a particularly well-structured linear system of equations. As such, our discussion will soon be translated into linear algebra. An important part of the structure comes from a probabilistic interpretation of the linear system.\textsuperscript{1}

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 16.1.** There exists a vector $x \in \Delta^V$ such that

$$x_v = \sum_{w:(u,v) \in E} \frac{x_u}{d^+(u)}$$ \hspace{1cm} (16.1)

for all $v \in V$. If $G$ is strongly connected, then this vector is unique and strictly positive.

\textsuperscript{1}Naturally, the mathematics we discuss existed long before search engines and similar ideas had been applied elsewhere.
16. Random walks

16.1. Ranking the web

To try to understand where such an $x$ comes from — and in particular, how we can assert that it describes a distribution over $B$ — consider the following random walk on $G$. At each step, you are on some vertex $u$. You choose an outgoing edge $(u, v) \in E$ uniformly at random, and step to $v$. This may send you walking chaotically all over the graph. On the world wide web, this is like randomly surfing the web where you keep following randomly chosen links. If the links tend to point to useful websites, then over time your random walk should stumble upon good web sites more often then a uniformly random sample of the web. Now, depending on where you start, after some number of $k$ steps, there are different probabilities of where you would end up. As $k$ increases, we might hope this reaches some kind of equilibrium, where the distribution is the same or close to the same between to $k$th and $(k + 1)$th step for sufficiently large $k$. This brings us to the notion of a stationary distribution.

**Definition 16.2.** 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 16.1, and consider the recursive relations satisfied by $x \in \Delta^V$:

$$x_v = \sum_{u: (u, v) \in E} \frac{x_u}{d^+(u)}$$

for all $v \in V$. In our random walk, from a given vertex $u$, we choose an outgoing edge $(u, v)$ with probability $1/d^+(u)$. In particular, if $u$ is chosen from a random distribution $x \in \Delta^V$, then the probability of then stepping to a particular vertex $v$ is

$$\sum_{u: (u, v) \in E} \frac{x_u}{d^+(u)}.$$

For the vector $x$ asserted by theorem 16.1, this sum equals $x_v$. That is, $x$ is the stationary distribution of a random walk on $G$. We can restate theorem 16.1 as follows.

**Theorem 16.1, restated.** Every random walk on a directed graph $G$ has a stationary distribution. If $G$ is strongly connected, then this distribution is unique.

16.1.1 PageRank

The actual PageRank algorithm proposed in [PBMW99] is slightly different. We augment the random walk thought 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 16.3. Let $G = (V, E)$ be a directed graph, and let $\alpha \in (0, 1)$. There exists a unique vector $x \in \Delta V$ such that for all $v \in V$,

$$x_v = (1 - \alpha) \sum_{u: (u,v) \in E} \frac{x_u}{d^+(u)} + \frac{\alpha}{n}.$$

The PageRank formulation has some additional convenient properties compared to general random walks. First, the vector $x$ is guaranteed to be unique, and has strictly positive coordinates. (Implicitly, it is the stationary distribution on the directed graph augmented by weighted edges between all pairs, which by theorem 16.1 is unique.) Second, it can be calculated more directly. We will come back to this point at the end of our discussion in section 16.6.

16.2 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. 16.3. 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

$$\text{score}_1(v) = \langle e_v, A1 \rangle = \sum_w (Ae_v)_w.$$  

We also have

$$d^+(u) = \langle 1, Ae_u \rangle.$$ 

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
\[
\text{score}_2(v) = \sum_{u : (u,v) \in E} \frac{1}{d^+(u)} = (AD^{-1} \mathbb{1})_v = \langle e_v, AD^{-1} \mathbb{1} \rangle.
\]

As for the stationary distribution ($\text{score}_3$), we see that the stationary distribution $x$ must satisfy the equation
\[
x = Rx \text{ for the linear map } R = AD^{-1} : \mathbb{R}^V \to \mathbb{R}^V.
\]

The map $R$ has the property that $Rx \in \Delta^V$ for any $x \in \Delta^V$. Any linear map $A : \mathbb{R}^n \to \mathbb{R}^n$ mapping $\Delta^n$ to $\Delta^n$ is called a stochastic linear function.

16.3 Eigenvectors

Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. An eigenvector of $A$ is a nonzero vector $x \neq 0$ such that
\[
Ax = \lambda x
\]
for some scalar value $\lambda \in \mathbb{C}$. The scalar $\lambda$ is called an eigenvalue of $A$, and here it is the eigenvalue corresponding to the eigenvector $x$.

Alternatively, a value $\lambda \in \mathbb{C}$ is an eigenvalue iff the linear map $(A - \lambda I) : \mathbb{R}^n \to \mathbb{R}^n$ is not invertible. Indeed, any eigenvector $x$ corresponding to $\lambda$ gives a second vector (besides $0$) such that $(A - \lambda I)x = 0$.

**Lemma 16.4.** 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 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.** A priori, it is not clear why every matrix should have an eigenvector.

**Lemma 16.5.** 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.

**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 16.6.** 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

\[
\langle x, (A^{-1})^T A^T y \rangle = \langle A^{-1} x, Ay \rangle = \langle AA^{-1} x, y \rangle = \langle x, y \rangle.
\]

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 \( (A^T)^T = A \).

**Lemma 16.7.** 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

\[
\text{rank}(A - \lambda I) = \text{rank}\left((A - \lambda I)^T\right) = \text{rank}(A - \lambda I).
\]
Lemma 16.8. 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 \( \langle x, y \rangle = 0 \).

Let us now restate theorem 16.1 in our new language of eigenvectors and eigenvalues.

**Theorem 16.1, 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.

16.4 The Perron-Frobenius theorem

**Definition 16.9.** 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 16.10.** Let \( A : \mathbb{R}^n \to \mathbb{R}^n \) be a positive linear map. 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 \( |\mu| < \lambda_1 \).

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 the supremum of \( L \) is the desired value \( \lambda_1 \). To this end, we first make the following claims about \( L \).

\(^2\)We will only prove that \( |\mu| \leq \lambda_1 \).
1. $L$ is nonempty.

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 (say) $Ax \geq \lambda x$ 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

\[
A\bar{x} = A \left( \lim_{n \to \infty} x_i \right) \overset{(d)}{=} \lim_{n \to \infty} Ax_i \geq \lim_{n \to \infty} \lambda_i x_i = \lambda \bar{x}.
\]

(d) invokes continuity of $A$ to pass through the limit.

We have now shown that $L$ is a closed and bounded set with at least one positive number. Any nonempty closed and bounded set $L$ has a finite supremum, $\lambda_1$, 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

\[
\frac{\epsilon}{1 - \epsilon} \leq \langle 1, y \rangle.
\]

Observe that

\[
(1 - \epsilon)x_1 + \frac{\epsilon}{\langle 1, y \rangle} \in \Delta^n.
\]

Moreover, we have

\[
A \left( (1 - \epsilon)x_1 + \frac{\epsilon}{\langle 1, y \rangle} y \right) \overset{(e)}{=} (1 - \epsilon)Ax_1 \overset{(d)}{=} (1 - \epsilon)\lambda(x_1 + y) \overset{(g)}{=} \lambda \left( (1 - \epsilon)x_1 + \frac{\epsilon}{\langle 1, y \rangle} y \right).
\]
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 $Ay = \lambda_1y$ for vector $y$. As mentioned above, we must have $y > 0$. If $y$ is not proportional to $x_1$, 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 $(A - \lambda_1I)^2y = 0$.

Then $(A - \lambda_1I)y$ is a simple eigenvector of $A$ with eigenvalue $\lambda_1$, so

$$Ay = \lambda_1y + cx_1$$

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 (2) 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 $\mu = x_1$.

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

$$\langle A^Tx, y \rangle = \langle x, Ay \rangle > 0.$$ 

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 $y_1$ 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 [Bel97; Lax07].
16.5 Perron-Frobenius for strongly connected random walks

We now extend theorem 16.10 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 16.11.** 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

$$A^T \mathbf{1} = \mathbf{1},$$

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 $\langle \mathbf{1}, x \rangle = 1$. We claim that $x \in \Delta^n$.

Consider the matrix $B = \frac{1}{n} \sum_{i=0}^{n-1} A^i$. 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.
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 16.10.
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 $\mathbb{1}$. So $\lambda_1 = 1$. Theorem 16.10 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 16.10 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.

### 16.6 Computing PageRank

The PageRank vector $x$ satisfies the equation

$$
x = (1 - \epsilon)Rx + \frac{\epsilon}{n} \mathbb{1},
$$

where $R = AD^{-1}$ models the random walk on the directed graph, and where $\mathbb{1} \in \mathbb{R}^V$ is the all-1’s vector. We can rewrite this as

$$
\left(\left(\frac{1}{1 - \epsilon} I - R\right)x = \frac{\epsilon}{(1 - \epsilon)n} \mathbb{1}.
$$

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

$$
x = \left(\frac{\epsilon}{(1 - \epsilon)n}\right) \left(\frac{1}{1 - \epsilon} I - R\right)^{-1} \mathbb{1} = \frac{\epsilon}{n} (I - (1 - \epsilon)R)^{-1} \mathbb{1}.
$$

We can write $(I - (1 - \epsilon)R)^{-1}$ as the infinite series

$$
(I - (1 - \epsilon)R)^{-1} = \lim_{k \to \infty} \sum_{i=0}^{k} ((1 - \epsilon)R)^i.
$$
Thus

\[ x = \frac{\epsilon}{(1 - \epsilon)n} \lim_{k \to \infty} \sum_{i=0}^{k} ((1 - \epsilon)R)^i \mathbb{1}. \]

The series on the RHS converges quickly for moderate \( \epsilon \), so in practice one only has to compute a few terms in the sum.

16.7 Additional notes and materials

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

16.8 Exercises

Exercise 16.1. Let \( G = (V, E) \) be a directed graph, not necessarily strongly connected. Recall that the strongly connected components 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 16.2. Let \( G = (V, E) \) be a simple\(^3\), unweighted, directed, and strongly connected graph. We proved in theorem 16.1 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.
17.1 Balanced coloring

Let $S_1, \ldots, S_m \subseteq [n]$ be a collection of $m$ sets of a universe of $n$ points. We would like to color all the points in one of two colors so that each set has the same number of points of each color, or as close to the same as possible.

More formally, we encode colorings as vectors $x \in \{-1, 1\}^n$ where $-1$ indicates one color and $+1$ indicates the other. For a set $S_i$, let

$$x(S_i) \overset{\text{def}}{=} \sum_{j \in S_i} x_j$$

denote the sum over the coordinates in $S_i$. The discrepancy of $S_i$ is defined as absolute
difference between the number of points of each color,

$$|x(S_i)| = \left| \sum_{j \in S_i} x_j \right|.$$  

The goal is to color the points as to minimize the maximum discrepancy:

$$\min_{x \in \{-1, 1\}^n} \max_{S_i} |x(S_i)|.$$  

The additive Chernoff bound and union bound imply that a uniformly random $z \in \{-1, 1\}^n$ has maximum discrepancy at most $O(\sqrt{n \log(m)})$. (Exercise 17.1.) 

Surprisingly, for the special case $m = n$, Spencer [Spe85] showed that there exists a coloring $z \in \{-1, 1\}^n$ with discrepancy

$$O(\sqrt{n}),$$

erasing the logarithmic factor. For $m > n$ the bound becomes

$$O\left(\sqrt{n \log(m/n)}\right)$$

However the proof was existential in nature and it remained an open to find such a coloring efficiently.

In 2010, Bansal [Ban10] discovered an algorithm that achieves discrepancy $O(\sqrt{n \log(m/n)})$; in particular this gives $O(\sqrt{n})$ for the symmetric setting $m = n$. Subsequent work by Lovett and Meka [LM12] gave an elegant randomized polynomial time algorithm achieving the more general bound of $O\left(\sqrt{n \log(m/n)}\right)$. This second algorithm is the main topic of this lecture. Formally, we will prove the following.

**Theorem 17.1.** Given $m$ sets $S_1, \ldots, S_m$ over $n$ points, there exists a vector $z \in \{-1, 1\}^n$ such that $z(S_i) \leq O\left(\sqrt{n \log(m/n)}\right)$ for all $i$. This vector $z$ can be computed in randomized polynomial time in expectation.

### 17.2 Reduction to partial coloring

A fractional coloring is defined as a vector $x \in [-1, 1]^n$. The algorithm maintains a fractional coloring $x \in [-1, 1]^n$, initially set to $x = 0$, that meets the discrepancy bound $|x(S_i)| \leq O\left(\sqrt{n \log(m/n)}\right)$ for all $i$, at all times. Progress is made by having the coordinates $x_j$ approach the integral extremes $\{-1, 1\}$ over time.
Let $\epsilon = 1/n$. We say that a coordinate $x_j$ is $\epsilon$-tight, or simply tight, if $|x_j| \geq 1 - \epsilon$. Otherwise we say that $x_j$ is free. We say that $x$ is tight if all the coordinates are tight. If $x$ is tight, then it is easy to see that we can deterministically round $x$ to a proper coloring in $\{-1, 1\}^n$ without significantly increasing the discrepancy. Thus the task is reduced to computing a tight fractional coloring $x$.

We break down this task further into partial coloring problems, where the goal is to obtain a fractional coloring $x$ with good discrepancy and where half the coordinates are tight. The tight coordinates are then fixed forever, and in the next iteration we apply the partial coloring procedure to the remaining free coordinates. Each iteration of partial coloring decreases the number of variables by half. The following lemma describes the guarantees of a partial coloring subroutine which we describe and analyze in the following section.

**Lemma 17.2.** Let $S_1, \ldots, S_m \subseteq [n]$, and let $x_0 \in [-1, 1]^n$. Let $\epsilon > 0$ be sufficiently small. Then there is a randomized algorithm that, in expected polynomial time, computes a point $x \in [-1, 1]^n$ such that

(i) $|x(S_i) - x_0(S_i)| \leq O\left(\sqrt{n \log(m/n)}\right)$ for all $j$.

(ii) At least half of the coordinates are tight.

We now explain at a high level how the partial coloring bounds of lemma 17.2 lead to a tight fractional covering with good discrepancy. Consider the first iteration. We apply partial-coloring and obtain a vector $x^{(1)}$ that (a) respects all of our desired discrepancy bounds and (b) makes at least half the variables tight. For all tight coordinates $i$, $x^{(1)}_i$ is forever fixed. Henceforth we restrict our attention to the remaining free variables $V^{(1)}$, which has half as many variables as before. Consider now the second iteration. We apply partial-coloring in $V^{(1)}$, now using $x^{(1)}$ (restricted to $V^{(1)}$) as the initial point $x_0$. Now we obtain a fractional coloring $x^{(2)}$ over $V^{(1)}$ where at least half the variables are tight, and where $x^{(2)} - x^{(1)}$ satisfies the desired discrepancy bound. This time, though, $n$ is divided by half, which improves the discrepancy bound:

$$|x^{(2)}(S_i) - x^{(1)}(S_i)| \leq O\left(\sqrt{n \log(m/n)}/2\right).$$

Continuing in this fashion, we repeatedly color and remove half the free variables. Each iteration pays a geometrically decreasing cost in the discrepancy, so the overall discrepancy for each set $S_i$ remains $O\left(\sqrt{n \log(m/n)}\right)$.

Pseudocode for this iterative algorithm is given in fig. 17.1. The following lemma and proof formalizes the high-level argument above.
iterative-partial-coloring($S_1,\ldots,S_m$)

1. Let $x^{(0)} = 0$ and $y^{(0)} = 0$.
2. For $t = 1, 2, \ldots$:
   A. Let $\mathcal{V}^{(t)} = \{ i : y_i^{(t-1)} \in (1-\epsilon, 1+\epsilon) \}$.
   B. If $\mathcal{V}^{(t)}$ is empty then return $y^{(t-1)}$.
   
   /* Otherwise partially color the loose vertices $\mathcal{V}^{(t)}$ with starting point given by
   $y^{(t-1)}$ restricted to $\mathcal{V}^{(t)}$. */
   C. Let $x^{(t)} = \text{partial-coloring}(S_1 \cap \mathcal{V}^{(t)}, \ldots, S_m \cap \mathcal{V}^{(t)}, \mathcal{V}^{(t)})$,
   D. Set $y_i^{(t)} = y_i^{(t-1)}$ for $i \notin \mathcal{V}^{(t)}$ and $y_i^{(t)} = x_i^{(t)}$ for $i \in \mathcal{V}^{(t)}$.

Figure 17.1: Iteratively applying the partial-coloring procedure to obtain a tight fractional coloring. See lemma 17.3.

**Lemma 17.3.** In expected polynomial time, iterative-partial-coloring returns a fractional coloring $y \in [-1, 1]^n$ such that:

(a) All coordinates $y_j$ are tight.
(b) All sets $S_i$ have $|y^{(t)}(S_i)| \leq O\left(\sqrt{n \log(m/n)}\right)$.

**Proof.** Let $T$ be the (random) number of iterations completed by iterative-partial-coloring; that is, iterative-partial-coloring returns $y^{(T)}$.

For each iteration $t$, we have

$$|y^{(t)}(S_i)| \leq |x^{(t)}(S_i \cap \mathcal{V}^{(t)}) - y^{(t-1)}(S_i \cap \mathcal{V}^{(t)})| + |y^{(t-1)}(S_i)|,$$

hence

$$|y^{(T)}(S_i)| \leq \sum_{t=1}^{T} |x^{(t)}(S_i \cap \mathcal{V}^{(t)}) - y^{(t-1)}(S_i \cap \mathcal{V}^{(t)})|.$$ We also have, for each iteration $t$,

$$|x^{(t)}(S_i \cap \mathcal{V}^{(t)}) - y^{(t-1)}(S_i \cap \mathcal{V}^{(t)})| \leq O\left(\sqrt{|\mathcal{V}^{(t)}| \log(m/|\mathcal{V}^{(t)}|)}\right).$$

Recall that $|\mathcal{V}^{(t)}| \leq n/2^t$, and observe that

$$\frac{n}{2^t} \log\left(\frac{m}{n/2^t}\right) = \frac{n}{2^t} \log(m/n) + \frac{n}{2^t} t \leq O\left(\frac{n}{1.9^t} \log(m/n)\right).$$
Plugging back in we have

\[ |y^{(T)}(S_i)| \leq O\left(\sqrt{n \log(m/n)}\right) \sum_{t=1}^T 1.9^{-t} = O\left(\sqrt{n \log(m/n)}\right), \]

as desired.

Now we are prepared to prove theorem 17.1, modulo the proof of lemma 17.2 which we prove next.

Let \( y \in [-1,1]^n \) be as described in lemma 17.3. Let \( z \in \{-1,1\}^n \) be the integral coloring obtained from \( y \) by deterministically setting \( z_i \in \{-1,1\} \) to the closer value to \( y_i \). This increases the discrepancy by at most \( \epsilon n \leq 1 \) for each set \( S_i \). This completes the proof.

### 17.3 Partial coloring by random walks: proof of lemma 17.2

#### The partial coloring algorithm

We now describe the algorithm underlying lemma 17.2. Let \( \lambda = c\sqrt{n \log(m/n)} \) for a sufficiently large constant \( c \). Let

\[ \mathcal{P} = \{ x \in [-1,1]^n : |x(S_i) - x_0(S_i)| \leq \lambda \text{ for all } S_i \}. \]

Our goal is to find a fractional coloring \( x \in \mathcal{P} \) where at least half the coordinates are tight.

We analyze an algorithm that can loosely be described as a high-dimensional random walk inside \( \mathcal{P} \). The algorithm starts at \( x^{(0)} = x_0 \). Each iteration we move from \( x^{(t-1)} \) to a random point \( x^{(t)} \in \mathcal{P} \) where the step \( x^{(t)} - x^{(t-1)} \) is sampled from a carefully chosen, high-dimensional Gaussian distribution.

As \( x^{(t)} \) moves randomly through \( \mathcal{P} \), \( x^{(t)} \) may approach some of the boundaries of \( \mathcal{P} \). For example, a coordinate \( x^{(t)}_j \) may become tight, or the discrepancy of a set \( S_i \) may approach its limit \( \lambda_i \). In this case the walk is constrained to avoid violating these critical constraints.

Towards a more formal description, let

\[ \delta = \epsilon / \text{poly}(m,n). \]

\( \delta \) controls average step size of the random walk. Let \( T = O(1/\delta^2) = \text{poly}(m,n)/\epsilon \). The algorithm runs for \( T \) iterations. We want to show that at termination, \( x^{(T)} \) is a tight coloring in \( \mathcal{P} \).
After $t$ iterations, we say that a coordinate $x_j^{(t)}$ is tight if $|x_j^{(t)}| \geq 1 - \epsilon$ (like before). We say that a set $S_i$ is tight if

$$|\langle S_i, x^{(t)} - x_0 \rangle| \geq \lambda - \epsilon.$$ 

The next random step, $x^{(t+1)} - x^{(t)}$, is restricted to ensure that the tight coordinates and sets are not further violated, as follows.

Let $S^{(t)}$ be the subspace orthogonal to all tight sets and coordinates with respect to $x^{(t)}$:

$$S^{(t)} \stackrel{\text{def}}{=} \{ y \in \mathbb{R}^n : y_j = 0 \text{ for all tight } x_j, y(S_i) = 0 \text{ for all tight } S_i \}.$$ 

Note that all the constraints are linear equations, so $S^{(t)}$ indeed describes a subspace. The step $x^{(t+1)} - x^{(t)}$ is sampled as a Gaussian vector from the subspace $S^{(t)}$.

To implement such a sample, let $k = \dim(S^{(t)})$, let $c_1, \ldots, c_k$ be an orthonormal basis for $S^{(t)}$, and let $U \in \mathbb{R}^{n \times k}$ be the matrix where the $i$th column is $c_i$. Let $g^{(t)} \sim \mathcal{N}^k$ be a vector of $k$ independent, normal Gaussian variables. We set

$$x^{(t+1)} = x^{(t)} + \delta U g^{(t)}.$$ 

Observe that since $U g^{(t)} \in S^{(t)}$, $x^{(t+1)}$ will have the same relationship with the variables and sets that were tight with respect to $x^{(t)}$; in particular, these tight inequalities will not get worse.

The following two lemmas help us understand the random vector $U g^{(t)}$.

**Lemma 17.4.** $\|Uw\|^2 = \|w\|^2$ for all $w \in \mathbb{R}^k$ and $\|U^Tv\|^2 \leq \|v\|^2$ for all $v \in \mathbb{R}^k$.

**Proof.** We have

$$Uw = \sum_{i=1}^{k} w_i c_i.$$ 

Since $c_1, \ldots, c_k$ is an orthonormal basis, we have

$$\|Uw\|^2 = \left( \sum_{i=1}^{k} w_i c_i, \sum_{i=1}^{k} w_i c_i \right) = \sum_{i=1}^{k} \sum_{j=1}^{k} w_i w_j \langle c_i, c_j \rangle = \sum_{i=1}^{k} w_i^2 = \|w\|^2.$$ 

For the second claim, observe

$$\|U^Tv\|^2 = \sum_{i=1}^{k} \langle c_i, v \rangle^2.$$ 

254
To interpret the RHS, extend $c_1, \ldots, c_k$ to an orthonormal basis $c_1, \ldots, c_n$ of $\mathbb{R}^n$. The vector $y \in \mathbb{R}^n$ defined by

$$y_i = \langle c_i, v \rangle \quad i = 1, \ldots, n$$

is simply a change of coordinates of $x$ to the new orthonormal basis, so

$$\|x\|^2 = \|y\|^2 = \sum_{i=1}^{n} \langle c_i, v \rangle^2.$$

Clearly the sum in the RHS is greater than the previous sum for $\|U^T v\|^2$.

**Lemma 17.5.** Let $k = \dim(S(t))$. For any vector $v$, $\langle v, U g(t) \rangle \sim \mathcal{N}(0, \sigma^2)$ where $\sigma^2 \leq \|v\|^2$.

**Proof.** Let $b = U^T v$. We have

$$\langle v, U g(t) \rangle = \langle U^T v, g(t) \rangle = \langle b, g(t) \rangle.$$

As we know, $\langle b, g(t) \rangle$ is distributed as a Gaussian with mean 0 and variance $\|b\|^2$. We have $\|b\|^2 = \|U^T v\|^2 \leq \|v\|^2$ be lemma 17.4.

In particular, lemma 17.5 implies that for each set $i$,

$$x(t+1)(S_i) - x(t)(S_i) = \delta \langle \mathbb{1}_{S_i}, U g \rangle$$

is distributed as a Gaussian with mean 0 and variance at most

$$\delta^2 |S_i| \leq \delta^2 n.$$

(Here $\mathbb{1}_{S_i}$ denotes the $\{0, 1\}$-indicator variable for $S_i$.) Lemma 17.5 also implies that each coordinate $j \in [n]$,

$$(x^{(t+1)} - x^{(t)})_j = \delta (Ug)_j$$

is distributed as a Gaussian with mean zero and variance at most $\delta^2$. Now recall that for a Guassian random variable $g$ with mean zero and variance $\sigma^2$,

$$\mathbb{P}[|g| \geq \lambda \sigma] \leq 2e^{-\lambda^2/2}$$

for all $\lambda$. Consequently we have

$$\mathbb{P}[|x^{(t+1)}(S_i) - x^{(t)}(S_i)| \geq \epsilon] \leq e^{-\epsilon^2/2\delta^2} = e^{-\text{poly}(m,n)}$$
for each \( S_i \) and similarly

\[
P \left[ \left| x_j^{(t+1)} - x_j^{(t)} \right| \geq \epsilon \right] \leq e^{-\text{poly}(m,n)}
\]

for each coordinate \( j \). Meanwhile the quantities are always 0 for tight constraints and tight coordinates because \( x^{(t+1)} - x^{(t)} \in S^{(t)} \).

Taking the union bound over all sets and variables, if \( x^{(t)} \in \mathcal{P} \), then with high probability, we have \( x^{(t+1)} \in \mathcal{P} \). Taking the union bound over all \( T \) iterations gives the following.

**Lemma 17.6.** With high probability, \( x^{(t)} \in \mathcal{P} \) for all \( t \in [T] \).

The remaining task is to show that at least half the variables are tight after \( T \) iterations.

**Lemma 17.7.** \( \mathbb{E} \left[ \|x^{(T)}\|^2 \right] \geq T \mathbb{E}[\text{dim}(S^{(T)})] \).

**Proof.** Each iteration \( t \), we have

\[
\|x^{(t)}\|^2 = \|x^{(t-1)}\|^2 + 2 \langle x^{(t-1)}, \delta U^{(t)} g^{(t)} \rangle + \delta^2 \|U^{(t)} g^{(t)}\|^2.
\]

When we take expectations, we have

\[
\mathbb{E}[\langle x^{(t-1)}, \delta U^{(t)} g^{(t)} \rangle] = \delta \mathbb{E}[\langle (U^{(t)})^T x^{(t-1)}, g^{(t)} \rangle] = 0
\]

because \( \langle (U^{(t)})^T x^{(t-1)}, g^{(t)} \rangle \) is a mean 0 Gaussian. We also have

\[
\mathbb{E}[\|U^{(t)} g^{(t)}\|^2] = \mathbb{E}[\|g^{(t)}\|^2] = \text{dim}(S^{(t-1)}).
\]

since \( \|g^{(t)}\|^2 \) is the sum of squares of \( \text{dim}(S^{(t)}) \) standard Gaussians. Thus

\[
\mathbb{E}[\|x^{(t)}\|^2 - \|x^{(t-1)}\|^2] = \delta^2 \mathbb{E}[\text{dim}(S^{(t-1)})].
\]

Unrolling, we have

\[
\mathbb{E}[\|x^{(T)}\|^2] \geq \sum_{t=1}^{T} \mathbb{E}[\|x^{(t)}\|^2 - \|x^{(t-1)}\|^2] \geq \sum_{t=1}^{T} \text{dim}(S^{(t-1)}).
\]

Finally, we observe that \( \text{dim}(S^{(t)}) \geq \text{dim}(S^{(T)}) \) since \( S^{(t)} \) only shrinks over time with additional tight constraints and variables.

\[
\square
\]

**Lemma 17.8.** \( \mathbb{E} \left[ \frac{\# \text{ tight coord.}}{\# \text{ tight sets}} \right] \geq (1 - \frac{1}{\delta^2 T}) n - \mathbb{E} \left[ \frac{\# \text{ tight sets}}{\# \text{ tight coord.}} \right] x^{(T)} \).
Proof. We first observe that
\[ E[\|x^{(T)}\|^2] \geq \delta^2 T \mathbb{E}[\dim(S^{(T)})] \geq T \left( n - \mathbb{E} \left[ \# \text{ tight sets } \frac{w}{r/t} x^{(T)} \right] - \mathbb{E} \left[ \# \text{ tight coord. } \frac{w}{r/t} x^{(T)} \right] \right) \]

We also claim that \( E[\|x^{(T)}\|^2] \leq n \). If so, then we would have
\[ n \geq E[\|x^{(T)}\|^2] \geq \delta^2 T \left( n - \mathbb{E} \left[ \# \text{ tight sets } \frac{w}{r/t} x^{(T)} \right] - \mathbb{E} \left[ \# \text{ tight coord. } \frac{w}{r/t} x^{(T)} \right] \right) , \]
which is the desired inequality up to rearrangement of terms.

To prove the claim, observe that for all coordinates \( j \), and all iterations \( t \), either
\[ \text{the } j\text{th coordinate is tight, or } \frac{E[(x^{(t)})_j^2 - (x^{(t-1)})_j^2]}{|(x^{(t-1)}_j| < 1 - \epsilon} \leq \delta^2. \]
It follows that \( E[(x^{(T)})_j^2] \leq 1 \) for all \( j \), hence \( E[\|x^{(T)}\|^2] \leq n \). \( \square \)

Lemma 17.9. \( E[\# \text{ tight sets } \frac{w}{r/t} x^{(T)}] \leq \frac{n}{4} \), hence \( E[\# \text{ tight coord. } \frac{w}{r/t} x^{(T)}] \geq \frac{3n}{4} \).

Proof. Fix a set \( S_i \). Let \( \mu = \lambda - \epsilon \); we again have \( \mu \geq c_1 \sqrt{n \log(m/n)} \) for an arbitrarily large constant \( c_1 \). We want to analyze the probability that \( |x^{(T)}(S_i) - x^{(0)}(S_i)| \geq \mu \).

For \( t \in [T] \), let
\[ Y_t = x^{(t)}(S_i) - x^{(t-1)}(S_i) = \delta U^{(t)} g^{(t)}. \]
Observe that
\[ x^{(T)}(S_i) - x^{(0)}(S_i) = Y_1 + \cdots + Y_T. \]
If they \( Y_t \)'s were independent, then we know the sum behaves like a Gaussian with variance at most \( T \delta^2 n \).

Here the \( Y_t \)'s are not independent: earlier iterations effect the variance of later iterations. However, each \( Y_t \) is a Gaussian conditional on \( Y_1, \ldots, Y_{t-1} \). In this case, one can still show that their sum will still behave as if they were independent; that is, as a Gaussian with variance at most \( T \delta^2 n \). (See lemma 17.11 in section 17.A.) Consequently
\[ \mathbb{P}[Y_1 + \cdots + Y_T \geq \mu] \leq 2e^{-\frac{\mu^2}{2T \delta^2 n}} \leq 2e^{-\frac{c_1^2 n \log(m/n)}{T \delta^2 n}} \leq \frac{n}{8m} \]
for a sufficiently large constant $c_1$.

Now, by linearity of expectation, the expected number of tight sets is
\[ E \left[ \text{# tight sets w/r/t } x(T) \right] = \sum_i P[S_i \text{ is tight}] \leq m \cdot \frac{n}{8m} = \frac{n}{8}, \]
as desired.

We now complete the proof by showing that at least half the coordinates are tight with constant probability. The preceding lemma shows that the expected number of free coordinates is at most $n/4$. By Markov’s inequality, there are at most $n/2$ free coordinates, hence at least $n/2$ tight coordinates, with probability at least $1/2$.

While we have only proven that partial-coloring succeeds with constant probability. Note that the guarantees of lemma 17.2 is easy to verify. This leads to a Las Vegas algorithm with expected polynomial running time that repeats the partial-coloring algorithm until it succeeds.

### 17.4 Additional notes and materials

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

### 17.5 Exercises

**Exercise 17.1.** Show that a uniformly random coloring of the $n$ points has maximum discrepancy at most $O(\sqrt{n \log m})$ with high probability.

**Exercise 17.2.** Suppose every set contains at most $L$ elements for some $L \in \mathbb{N}$. Adjust the $O(\sqrt{n \log(m/n)})$-discrepancy algorithm of this chapter to obtain $O(\sqrt{L \log(m/n)})$-discrepancy $O(\sqrt{L \log(m/n) \log(n)})$ in this setting.\(^1\) Here, rather than repeat the entire analysis verbatim, explain where $L$ enters the analysis and how it fits in and propagates through the rest of the proof to obtain the desired bound. (At least one critical calculation should be redone.)

\(^1\)In particular, show that lemma 17.2 holds with discrepancy $O\left(\sqrt{L \log(m/n)}\right)$. The additional $\log(n)$ comes from running partial coloring $O(\log(n))$ times.
17.A Analyzing Gaussian martingales

The proof of lemma 17.9 involves analyzing a sequence sum of random variables where each term, conditional on all previous terms, was distributed as a Gaussian. There we suggested that the sum behaves similarly as if it were a sum of independent Gaussians. Here we provide the formal details.

Lemma 17.10. Let $X$ be a Gaussian random variable with mean 0 and variance $\sigma^2$. Then for all $t \in \mathbb{R}$.

$$E[e^{tX}] = e^{t^2 \sigma^2 / 2}.$$  

Proof. WLOG we may assume $\sigma = 1$. Let $f(x) = e^{-x^2 / 2}$ denote the density function of $X$. We have

$$E[e^{tX}] = \int e^{tx} f(x) \, dx = \int f(x-t) \, dx = e^{t^2 / 2} f(x-t),$$

where (a) observes that

$$e^{tx} f(x) = \frac{1}{\sqrt{2\pi}} e^{tx - x^2 / 2} = \frac{e^{t^2 / 2}}{\sqrt{2\pi}} e^{-(x-t)^2 / 2} = e^{t^2 / 2} f(x-t).$$

Lemma 17.11. Let $Y_1, \ldots, Y_n \in \mathbb{R}$ be random variables where for each $i \in [n]$, conditional on $Y_1, \ldots, Y_{i-1}$, $Y_i$ is a Gaussian with mean 0 and variance at most $\sigma^2$. Then

$$P[Y_1 + \cdots + Y_n \geq \lambda] \leq e^{-\lambda^2 / 2n\sigma^2}.$$  

Proof. Let $t \in \mathbb{R}$ be a parameter TBD. By exponentiating and taking Markov’s inequality (as usual), we have

$$P[Y_1 + \cdots + Y_n \geq \lambda] \leq E[e^{tY_1 + \cdots + Y_n}] e^{-t\lambda} = \prod_{i=1}^{n} E[e^{tY_i} \mid Y_1, \ldots, Y_{i-1}] e^{-t\lambda}$$

For each $i$, by lemma 17.10, we have

$$E[e^{tY_i} \mid Y_1, \ldots, Y_{i-1}] \leq e^{t^2 / 2\sigma^2}.$$  

Plugging back in, we have

$$P[Y_1 + \cdots + Y_n \geq \lambda] \leq e^{nt^2 / 2\sigma^2 - t\lambda}.$$  

The RHS is minimized by $t = \lambda / n\sigma^2$, which gives

$$P[Y_1 + \cdots + Y_n \geq \lambda] \leq e^{-\lambda^2 / 2n\sigma^2},$$

as desired. □

259
Chapter 18

Connectivity and Electricity

18.1 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 [Sav70]. Now, can $(s, t)$-connectivity by solved in $O(\log n)$ space? This is basically just enough space to keep track of what vertex you are currently on as you search a graph.

Consider the following simple algorithm. Given $s, t \in V$, start a random walk from $s$, where in each iteration you pick a random neighbor of the current vertex and move there. Randomly walk for $O(mn)$ steps. If you come across $t$ at any point, then you know that $s$ and $t$ are connected. If not, then you answer no.

Clearly the above algorithm takes only $O(\log n)$ space. You only need to keep
track of which vertex you are on, and generate a random number between 1 and (at most) \( n \) to randomly select a neighbor. You also need to count the number of steps you’ve taken so far. When the algorithm answers in the affirmative, you are always right. Unfortunately, if it answers no, the algorithm could be wrong. Certainly there is no proof that there is no path from \( s \) to \( t \). But today we will show that the algorithm is correct with constant probability. Therefore you can rerun the experiment \( O(\log n) \) times to be correct with high probability.

To frame the analysis we introduce the following definition.

**Definition 18.1.** 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 18.2.** 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:

\[
\text{Can (s, t)-connectivity be decided deterministically in } O(\log n) \text{ space?}
\]

We will return to this question later in chapter 21.

### 18.2 Electrical networks

Our analysis will be based on a (perhaps surprising) connection between random walks and electrical networks. For the sake of our discussion, an electrical network is an undirected graph \( G = (V, E) \) with positive edge weights \( r : E \to \mathbb{R}_{>0} \) called resistances. If one attaches a battery to two vertices \( s \) and \( t \), it induces a current that (in our discussion) is a unit flow from \( s \) to \( t \). As the electricity flows from \( s \) to \( t \), it is said to take the path of least resistance. What is the path of least resistance? A computer scientist or operations researcher might suggest this is the shortest path from \( s \) to \( t \) with respect to 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:

$$\minimize (f, Rf) = \sum_{e \in E} r_e f_e^2 \text{ over } f \in \mathbb{R}^E \text{ s.t. } Bf = d.$$

(18.1)

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 (18.1).

2. The second part is about interpreting hitting times and cover times via electrical networks, and proving the desired bounds.

18.3 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.

18.3.1 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 18.3.** 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 an 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

\[
A^T y = \varphi'(x).
\]
18.3.2 Ohm’s Law

If we apply theorem 18.3 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 18.4.** 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 18.3, 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

\[
\langle g, Rg \rangle - \langle f, Rf \rangle \stackrel{(a)}{=} 2\langle Rf, g - f \rangle \stackrel{(b)}{=} 2\langle B^Tp, g - f \rangle = 2\langle p, B(g - f) \rangle = 2\langle p, d - d \rangle = 0.
\]

Here (a) applies convexity of \( \varphi(x) \), and (b) substitutes \( r = R^{-1}Bp \). \qed

18.4 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

\[
\langle x, Lx \rangle = \sum_{e = \{u, v\} \in E} w(e)(x_u - x_v)^2.
\]

**Lemma 18.5.** Let \( w(e) = 1/r_e \) for all \( e \). Then \( L = BR^{-1}B^T \).

To prove lemma 18.5, 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 18.3.

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., \( 1 \).
Theorem 18.6. 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.

18.5 Effective conductance

Consider the following optimization problem.

\[
\text{minimize } \langle p, Lp \rangle \text{ over } \langle p, d \rangle = 1. \tag{18.2}
\]

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 (18.2) 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 (18.2), 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 18.7. (18.2) has optimum solution $p^* = L^{-1}d/2\langle d, L^{-1}d \rangle$ 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 18.8. 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 18.9. 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).

18.6 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 18.1. 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 18.10.** 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 18.11.** 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

$$H(u, v) + H(v, u) = (p_u - p_v) + (q_v - q_u) = r_u - r_v$$

268
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

\[
H(s, t) + H(t, s) = \langle r, d \rangle = \frac{1}{2m} \langle r, B B^T r \rangle \\
= \frac{1}{2m} \langle B^T r, B^T r \rangle = 2m \langle (1/2m)B r, (1/2m)B r \rangle,
\]

as desired.

\[\square\]

**Theorem 18.12.** 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 18.11 as an upper bound on the effective resistance gives the bound we seek. \[\square\]

**Theorem 18.2.** 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 \( C(s) \leq m(n - 1) \).
18. Connectivity and Electricity

18.7. Additional notes and materials

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

$$\sum_{e = \{u, v\} \in T} H(u, v) + H(v, u) \leq \sum_{e = \{u, v\} \in T} 2m \leq 3m(n - 1).$$

\[ \square \]

18.7 Additional notes and materials

See also [DS84].

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

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

18.8 Exercises

Exercise 18.1. 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 18.2. 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 18.3. Complete the proof of lemma 18.5.
Chapter 19

Spectral analysis of undirected random walks

19.1 The Laplacian of a graph

Let $G = (V, E)$ be an undirected graph with positive edge weights $w \in \mathbb{R}^E > 0$. In our analysis of electrical networks (chapter 18), we briefly came across the Laplacian form of $G$. We reintroduce the Laplacian $L : \mathbb{R}^V \to \mathbb{R}^V$ of from a different perspective.

We first show how to model a single edge by a rank-1 matrix; an entire graph is then modeled by the corresponding weighted sum over its edges. We will work in the $n$-dimensional vector space $\mathbb{R}^V$ – one coordinate per vertex. Edges are modeled as matrices in $\mathbb{R}^{V \times V}$.

The Laplacian of an (unweighted) edge $e = \{u, v\}$ is the rank-1 matrix

$$L_e = (1_u - 1_v) \otimes (1_u - 1_v)$$

where $1_u \in \{0, 1\}^V$ denotes the indicator vector\(^1\) for $u$. Here $a \otimes b$ denotes the outer product of two vectors $a, b$, defined by $\langle x, (a \otimes b)y \rangle = \langle a, x \rangle \langle b, y \rangle$. Note that the expression for $L_e$ is indifferent to whether we wrote $1_u - 1_v$ or $1_v - 1_u$, as long as it is symmetric. For any input vector $x \in \mathbb{R}^V$, we have

$$\langle x, L_e x \rangle = (x_u - x_v)^2.$$

For an undirected graph $G = (V, E)$ with positive edge weights $w : E \to \mathbb{R}_{>0}$, the Laplacian of the graph is the weighted sum of Laplacians of its edges,

$$L = \sum_e w(e)L_e.$$

Given an input vector $x \in \mathbb{R}^V$, we have

$$\langle x, Lx \rangle = \sum_{e \in E} w(e)\langle x, L_e x \rangle = \sum_{e = (u,v) \in E} w(e)(x_u - x_v)^2.$$

\(^1\)We are avoiding the conventional notation $e_u$ for the standard basis vectors because $e$ is so frequently used for edges.
That is, the $L$ induces a simple sum of squared differences on $x$, based on the edges of the graph.

In fact, it induces a very familiar sum of squares. Recall that the 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 $f = R^{-1}B^T p$ for a set of vertex potentials $p$. Then we have

$$\langle f, Rf \rangle = \langle R^{-1}B^T p, RR^{-1}B^T p \rangle = \sum_{e=(u,v) \in E} \frac{1}{r_e} (p_u - p_v)^2.$$ 

That is, we are choosing $p$ as to minimize the Laplacian of the graph with edge weights $w(e) = 1/r_e$.

The Laplacian $L$ is also closely tied to the cuts of $G$. Given a set $S \subset V$, if we letting $1_S$ and $1_{\bar{S}}$ denote the indicator vectors of $S$ and $\bar{S}$, we have

$$\langle 1_S, L1_{\bar{S}} \rangle = 4w(\delta(S)).$$

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,

$$\langle x, Lx \rangle \geq 0$$

for all $x \in \mathbb{R}^n$.

These two properties make $L$ a member of the following very importance class of linear operators.

**Definition 19.1.** A linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is a positive semi-definite linear operator if

(a) $A$ is symmetric.

(b) $\langle x, Ax \rangle \geq 0$ for all $x \in \mathbb{R}^n$.

A is (strictly) positive definite if in addition to being positive semi-definite,

(c) $A$ is invertible.

The Laplacian $L$ is not invertible: $L1 = 0$. 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 exercise 19.3).
19. Spectral analysis of undirected random walks

19.2. The Spectral Theorem for Symmetric Maps

Kent Quanrud
Fall 2022

19.2 The Spectral Theorem for Symmetric Maps

Our first discussion on random walks (chapter 16) showed that eigenvectors give insight into the behavior of a linear map. We will see that the eigenvectors of symmetric linear maps — such as the Laplacian — are particularly well-behaved and useful.

Let \( A : \mathbb{R}^n \rightarrow \mathbb{R}^n \) be a linear map. Recall that a vector \( x \in \mathbb{R}^n \) is an eigenvector of \( A \) with eigenvalue \( \mu_1 \in \mathbb{C} \) of \( Ax = \mu_1 x \). Recall the following facts, which apply generally to all linear maps.

**Fact 19.2.** Let \( A : \mathbb{R}^n \rightarrow \mathbb{R}^n \) be a linear map. Then \( A \) has an eigenvalue \( \mu_1 \in \mathbb{C} \) and eigenvector \( x \in \mathbb{C}^n \).

If \( A \) is symmetric, then we can strengthen this fact to assert a real-valued eigenvalue and eigenvector with real-valued coordinates, and moreover they can be obtained by optimization.

**Lemma 19.3.** Let \( L : X \rightarrow X \) be a symmetric linear map in a vector space \( X \) over \( \mathbb{R} \). Let \( x \) maximize \( \langle x, Lx \rangle \) subject to \( \| x \| = 1 \). Then \( Lx = \mu_1 x \) for \( \mu_1 = \langle x, Lx \rangle \).

**Proof.** We claim that for any \( u \in X \) with \( \| u \| = 1 \) and \( \langle u, x \rangle = 0 \), \( \langle u, Lx \rangle = 0 \). If \( Lx \) is orthogonal to \( u \) for every \( u \) orthogonal to \( x \), then we must have \( Lx \in \text{span}(x) \); i.e., \( Lx = \mu_1 x \) for some \( \mu_1 \in \mathbb{R} \). Upon inspection, \( \mu_1 = \mu_1 \langle x, x \rangle = \langle x, Lx \rangle \), as claimed.

Let \( u \in X \) with \( \| u \| = 1 \) and \( \langle u, x \rangle = 0 \). Define

\[
f(\epsilon) = \left( \frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}}, L \left( \frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}} \right) \right) = \frac{\langle x + \epsilon u, L(x + \epsilon u) \rangle}{1 + \epsilon^2}.
\]

\( f(\epsilon) \) 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 20.2.)

**Remark 19.4.** An alternative proof starts from the fact there exists a complex eigenvalue and eigenvector, and goes on to show that this eigenvalue must be real-valued and that there is a corresponding eigenvector with real-valued coordinates.

The following theorem, called the *spectral theorem* for symmetric operators, strengthens the previous lemma to show that all the eigenvalues and eigenvectors are real-valued.
Theorem 19.5. Let $X$ be an $n$-dimensional vector space over $\mathbb{R}$. Let $A : X \to X$ be a symmetric linear map. Then there exists an orthonormal basis $v_1, \ldots, v_n$ of $X$ and $n$ scalar values $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ such that

$$A = \lambda_1(v_1 \otimes v_1) + \lambda_2(v_2 \otimes v_2) + \cdots + \lambda_n(v_n \otimes v_n).$$

(19.1)

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 19.3, $A$ has a real-valued eigenvalue $\mu_1$ with a corresponding eigenvector $u \in X$. By scaling $u$, we may assume $\|u\| = 1$. Consider the map $B = A - \mu_1(u \otimes u)$. $B$ is also symmetric, and maps the space $\text{span}(x) = \{\alpha x : \alpha \in \mathbb{R}\}$ 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 $v_1, \ldots, v_{n-1}$ of $Y$ and scalar values $\lambda_1, \ldots, \lambda_{n-1} \in \mathbb{R}$ such that

$$B = \lambda_1(v_1 \otimes v_1) + \cdots + \lambda_{n-1}(v_{n-1} \otimes v_{n-1}).$$

Let $\lambda_n = \mu_1$ and $v_n = u$. Observe that $v_1, \ldots, v_n$ is an orthonormal basis of $X$. We have

$$\lambda_1(v_1 \otimes v_1) + \cdots + \lambda_n(v_n \otimes v_n) = B + \lambda_n(v_n \otimes v_n) = A,$$

as desired. \hfill \Box

Theorem 19.5 makes the structure of any symmetric map $A : \mathbb{R}^n \to \mathbb{R}^n$ extremely transparent. By theorem 19.5, let $v_1, \ldots, v_n \in \mathbb{R}^n$ and $\mu_1, \ldots, \mu_n \in \mathbb{R}$ be such that

$$A = \mu_1(v_1 \otimes v_1) + \cdots + \mu_n(v_n \otimes v_n).$$

We assume that the $\mu_i$’s are in nonincreasing order: $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n$.\footnote{Notationally, we try to use $\mu_i$’s when listing the eigenvalues in decreasing order, and $\lambda_i$’s when listing the eigenvalues in increasing order.}

For any input vector $x \in \mathbb{R}^n$, we can write $x$ uniquely in the basis $\{v_1, \ldots, v_n\}$ as

$$x = \alpha_1 v_1 + \cdots + \alpha_n v_n,$$
where $\alpha_i = \langle x, v_i \rangle$. Then we have

$$Ax = \mu_1 \alpha v_1 + \cdots + \mu_n \alpha_n v_n.$$  

That is, in the basis $\{v_1, \ldots, v_n\}$, $A$ simply rescales the $i$th coordinate by a factor of $\mu_i$. That is to say:

*Every symmetric matrix is a diagonal matrix up to rotation (i.e., change in basis).*

We can see from the construction in the proof that the $\mu_i$’s are the eigenvalues of $A$ and the $v_i$’s are eigenvectors. But this fact is even more obvious in hindsight given the spectral representation (19.1). For each $i$, we have

$$Av_i = \mu_i (v_i \otimes v_i) v_i = \mu_i v_i,$$

by orthonormality of the $v_i$’s. The following theorem gives a min-max characterization of the eigenvalues and follows immediately from the spectral theorem. It is often called the Courant-Fischer minimax theorem.

**Theorem 19.6.** Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a symmetric linear operator. Let $\mu_1, \ldots, \mu_n$ be the $n$ eigenvalues of $A$ (with multiplicity) in decreasing order. Then

$$\mu_k = \min_{S, \text{dim}(S) = k-1} \max_{x \in X/S} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.$$  

### 19.3 Random walks in undirected graphs

Let $G = (V, E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let us assume that $G$ is connected. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map of $G$. Recall that $R = AD^{-1}$, where $A : \mathbb{R}^V \to \mathbb{R}^V$ is the weighted adjacency map and $D = \text{diag}(A1)$ 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 1 that defines a unique stationary distribution. Before, we proved the existence of a unique stationary distribution for strongly connected directed random walk. For undirected random walk, the stationary distribution is very straightforward.

**Theorem 19.7.** Let $G = (V, E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map of $G$. Then the stationary distribution of $R$ is proportional to the weighted degrees of its vertices.
Proof. We already know that the stationary distribution exists and is unique. We have
\[ R(A1) = A(\text{diag}(A1))^{-1}A1 = A1. \]

While \( R \) is not a symmetric map\(^3\), we can extend the spectral theorem for symmetric maps to \( R \) by way of similarity.

**Definition 19.8.** Two linear maps \( A, B : X \to X \) are similar if \( A = C^{-1}BC \) for an invertible map \( C : X \to X \).

**Lemma 19.9.** Let \( A \) and \( B \) be similar. Then their kernels are isomorphic. In particular, if \( A = C^{-1}BC \), then \( C \) restricts to an r isomorphism between \( \ker(A) \) and \( \ker(B) \).

**Lemma 19.10.** Let \( A, B : X \to X \) be two linear maps. If \( A \) and \( B \) are similar, then \( A \) and \( B \) have the same eigenvalues with the same multiplicities. If \( A = C^{-1}BC \), then \( C \) maps eigenvectors of \( A \) to eigenvectors of \( B \) with the same eigenvalues.

Proof. For all \( \lambda \), \( A - \lambda I \) and \( B - \lambda I \) are also similar. \( \square \)

We introduce the normalized walk matrix as the map \( Q : \mathbb{R}V \to \mathbb{R}V \) defined by
\[ Q = D^{-1/2}RD^{1/2} = D^{-1/2}AD^{-1/2}. \]

By the first equality above, \( Q \) is similar to \( R \), and thus has all its eigenvalues in the range \([-1, 1]\) and eigenvalue 1 with multiplicity 1. On the other hand, by the second equality, \( Q \) is symmetric. As such, it has an orthonormal basis of eigenvectors. Let \( 1 = \mu_1, \ldots, \mu_n \geq -1 \) list the eigenvalues of \( Q \) in decreasing order. Let \( u_1, \ldots, u_n \) be an orthonormal basis of \( \mathbb{R}V \) such that
\[ Q = u_1 \otimes u_1 + \mu_2(u_2 \otimes u_2) + \cdots + \mu_n(u_n \otimes u_n). \]

(Here we substituted \( \mu_1 = 1 \)). We also know, from ??, that \( D^{1/2}u_1 \) must correspond to the uniform distribution, \( d/\langle 1, d \rangle \). Since \( u_1 \) has unit length, we have
\[ u_1 = \frac{D^{-1/2}(d)}{\|D^{-1/2}d\|} = \frac{1}{\sqrt{2W}}\sqrt{d}, \]

---

\(^3\)unless \( G \) is regular; see exercise 19.5
where \( \sqrt{d} \) represents the entrywise square root of \( d \), and \( W = \sum_{e \in E} w(e) = \frac{1}{2} \sum_{v \in V} d(v) \) is the sum of all edge weights. Substituting back in, we have

\[
Q = \frac{1}{2W} \left( \sqrt{d} \otimes \sqrt{d} \right) + \mu_2(u_2 \otimes u_2) + \cdots + \mu_n(u_n \otimes u_n).
\]

Let us now consider the convergence rate of a random walk. Let \( x \in \Delta^V \) be any initial probability distribution over \( Q \). We want to understand the distribution \( R_k x \) obtained after \( k \) steps of the random walk. Observe first that

\[
R_k x \overset{(a)}{=} D^{1/2} Q^k D^{-1/2} x
\]

\[
= D^{1/2} \left( \frac{1}{2W} \left( \sqrt{d} \otimes \sqrt{d} \right) + \mu_2(u_2 \otimes u_2) + \cdots + \mu_n(u_n \otimes u_n) \right)^k D^{-1/2} x
\]

\[
\overset{(b)}{=} D^{1/2} \left( \frac{1}{2W} \left( \sqrt{d} \otimes \sqrt{d} \right) + \mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n) \right) D^{-1/2} x
\]

\[
\overset{(c)}{=} \frac{d}{2W} + \left( \mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n) \right) D^{-1/2} x
\]

(a) substitutes in \( R = D^{1/2} Q D^{-1/2} \), where the \( D^{-1/2} \) and \( D^{1/2} \) terms between \( Q \)'s cancel out. (b) is because the \( u_i \)'s are othonormal \(^4\) – here we see some of the power of the spectral theorem! (c) is because

\[
D^{1/2} \left( \sqrt{d} \otimes \sqrt{d} \right) D^{-1/2} x = (d \otimes \mathbb{1}) x = \langle \mathbb{1}, x \rangle d = d.
\]

Consider the RHS of the last equation above. Remarkably, the stationary distribution, \( d/2W \), has emerged, followed by a messy term involving the non-dominant eigenvalues and eigenvectors. Thus the difference between \( R_k x \) and the stationary distribution is precisely

\[
D^{1/2} \left( \mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n) \right) D^{-1/2} x.
\]

Let \( S = \mu_2^k(u_2 \otimes u_2) + \cdots + \mu_n^k(u_n \otimes u_n) \); \( S \) is symmetric, with eigenvalues \( 0, \mu_2^k, \ldots, \mu_n^k \).

\(^4\)We should point out that \( (a \otimes b)(c \otimes d) = \langle b, c \rangle (a \otimes d) \)
Let $\Delta_{\text{max}}$ be the maximum degree in $G$ and let $\Delta_{\text{min}}$.

\[
\left\| R^k x - \frac{1}{2W} d \right\|^2 = \left\| D^{1/2} S D^{-1/2} x \right\|^2 \\
= \left\langle S D^{-1/2} x, D \left( S D^{-1/2} x \right) \right\rangle \\
\overset{(d)}{\leq} \Delta_{\text{max}} \left\| S D^{-1/2} x \right\|^2 \\
= \Delta_{\text{max}} \left\langle D^{-1/2} x, S^2 \left( D^{-1/2} x \right) \right\rangle \\
\overset{(e)}{\leq} \max \left\{ \mu_2^k, \mu_n^k \right\} \Delta_{\text{max}} \left\langle x, D^{-1} x \right\rangle \\
\leq \max \left\{ \mu_2^k, \mu_n^k \right\} \frac{\Delta_{\text{max}}}{\Delta_{\text{min}}}.
\]

(d) and (e) both follow from the fact that for a symmetric map $A$ with maximum eigenvalue $\mu_1$, we have $\langle x, Ax \rangle \leq \mu_1 \|x\|^2$ for all $x$ (lemma 19.3). $D$ has maximum eigenvalue $\Delta_{\text{max}}$, $S^3$ has maximum eigenvalue $\max \left\{ \mu_2^k, \mu_n^k \right\}$, and $D^{-1}$ has maximum eigenvalue $1/\Delta_{\text{min}}$. Recall that $\mu_2, \mu_n \in [-1, 1]$. If $\mu_2$ and $\mu_n$ are both bounded away from both 1 and $-1$, then $\max \left\{ \mu_2^k, \mu_n^k \right\} = \max \left\{ \mu_2, |\mu_n| \right\}^k \to 0$ as $k \to \infty$. To this end, the spectral gap of a random walk $R$ is defined as the difference

\[
\gamma = 1 - \max \left\{ \mu_2, |\mu_n| \right\},
\]

where $\mu_2$ is the second largest eigenvalue and $\mu_n$ is the smallest eigenvalue. We have given the following bound on the convergence rate as a function of the spectral gap.

**Theorem 19.11.** Let $G = (V, E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $G$ be connected. Let $d \in \mathbb{R}_{>0}^V$ be the weighted degrees of the vertices. Let $\Delta_{\text{max}} = \max_v d(v)$ be the maximum weighted degree and let $\Delta_{\text{min}} = \min_v d(v)$ be the minimum weighted degree. Let $W = \sum_{e \in E} w(e)$ be the sum of edge weights. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map of $G$ and let $\gamma$ 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

\[
\left\| R^k x - s \right\| \leq (1 - \gamma)^k \sqrt{\frac{\Delta_{\text{max}}}{\Delta_{\text{min}}}}.
\]
19. Spectral analysis of undirected random walks

19.5 Exercises

Exercise 19.1. Let $A \in \mathbb{R}^{n \times n}$. Prove or disprove: $A$ is symmetric iff $\langle x, Ax \rangle = \langle x, A^T x \rangle$ for all $x$.

Exercise 19.2. Let $A \in \mathbb{R}^{n \times n}$ be positive semi-definite. Prove or disprove: $A$ is positive definite iff $\langle x, Ax \rangle > 0$ for all $x \neq 0$.

Exercise 19.3. Let $G = (V, E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of $G$. Prove that $G$ is connected iff for any $x \not\in \text{span}(\mathbb{1})$, we have $\langle x, Lx \rangle > 0$.

Exercise 19.4. Finish the proof of lemma 19.3, 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 19.5. Let $G = (V, E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $R : \mathbb{R}^V \to \mathbb{R}^V$ be the random walk map. Prove that $R$ is symmetric iff $G$ is regular.

Exercise 19.6. Suppose your goal is to start at a random walk at a single vertex and converge to the stationary distribution as fast as possible. Show that one can choose a vertex $v$ such that, starting from an initial distribution of $x = 1_v$, the $\ell_2$-distance from the stationary distribution after $k$ steps is at most $(1 - \gamma)^k$.

Exercise 19.7. 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 $G$. 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 $G$.

---

5A graph is regular if every vertex has the same weighted degree
Chapter 20

Conductance

20.1 Sparse cuts

Recall that the sparsity of a cut \( \delta(S) \) (where \( S \subset V \)), which we denote \( \Phi(S) \), was the ratio

\[
\Phi(S) \overset{\text{def}}{=} \frac{\bar{w}(\delta(S))}{\min\{|S|,|\bar{S}|\}},
\]

where \( \bar{S} = V \setminus S \). The sparsity of the graph \( G \) is defined as the sparsity of the sparsest cut,

\[
\Phi(G) \overset{\text{def}}{=} \min_{S \subset V} \Phi(S).
\]

To relate sparsity to the Laplacian, note that for any nonempty set \( S \subset V \) with at most \( n/2 \) vertices, we have

\[
\Phi(S) = \frac{\bar{w}(\delta(S))}{|S|} = \frac{\langle 1_S, L 1_S \rangle}{\langle 1_S, 1_S \rangle},
\]

(20.1)

where \( 1_S \) 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,

\[
x = 1_S - \alpha 1 \text{ where } \alpha = \langle 1_S, 1 \rangle / \langle 1, 1 \rangle = |S|/n
\]
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 20.1.** Let \( G = (V, E) \) be an undirected graph with \( m \) edges and \( n \) vertices, and positive edge weights \( w : E \to \mathbb{R}_{>0} \). Let \( L : \mathbb{R}^V \to \mathbb{R}^V \) be the Laplacian of \( G \). Let \( \lambda_2 \) be the second smallest eigenvalue of \( L \) and let \( \Phi(G) \) be the sparsity of \( G \). Then
\[ \lambda_2 \leq n\Phi(G). \]
20. Conductance

20.2. Conductance

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. 20.1 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 (20.1). While the numerator in (20.1) seems appropriate, the denominator does not capture the volume. Instead, consider the following quotient:

$$\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} \text{ where } x \in \mathbb{R}^V.$$ (20.2)

For any set $S$ with at most half the total volume, we have

$$\Psi(S) = \frac{\bar{w}(\delta(S))}{\text{vol}(S)} = \frac{\langle 1_S, L1_S \rangle}{\langle 1_S, D1_S \rangle}.$$ (20.2)

That said, the quotient (20.2) 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

$$M = D^{-1/2}LD^{-1/2}.$$ (20.2)

For any vector $x$, letting $y = D^{1/2}x$, we have

$$\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} = \frac{\langle y, My \rangle}{\langle y, y \rangle}.$$ (20.2)

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 \( R = AD^{-1} : \mathbb{R}^V \rightarrow \mathbb{R}^V \) 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 20.2.** 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 \( M : \mathbb{R}^V \rightarrow \mathbb{R}^V \) be the normalized Laplacian and \( R : \mathbb{R}^V \rightarrow \mathbb{R}^V \) 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 20.3.** 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 20.3 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 20.3. We will prove the harder inequality momentarily in section 20.3.

**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 \( M \).
20. Conductance

20.3 Proving $\Psi(G) \leq \sqrt{2\lambda_2}$

- 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 it this value is very close to $-1$. So instead we analyze the closely related lazy random walk where this exception does not occur.

**Theorem 20.4.** 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

$$\|S^t x - s\| \leq \exp\left(-t\Psi^2/4\right)\sqrt{\frac{\Delta_{\max}}{\Delta_{\min}}}$$

for any initial distribution $x$.

**Proof.** Let $S$ denote the lazy random walk matrix. We have $S = I/2 + R/2$, hence $\mu_k(S) = 1/2 + \mu_k(R)/2$ for all $k$.\footnote{Here $\mu_k(A)$ denotes the $k$th largest eigenvalue of $A$, and $\lambda_k(A)$ denotes the $k$th smallest eigenvalue of $A$.} In particular, since $\mu_n(S) \geq -1/2$, $S$ has spectral gap

$$\gamma_S = \min\{1 - \mu_2(S), 1 + \mu_n(S)\} \geq \frac{1}{2} \min\{1 - \mu_2(R), 1\}.$$ 

Furthermore, by similarity of $I - R$ and $M$, followed by Cheeger’s inequality, we have

$$1 - \mu_2(R) = \lambda_2(I - R) = \lambda_2(M) \geq \Psi^2/2,$$

hence $\gamma_S \geq \min\{\Psi^2/2, 1\}/2 = \Psi^2/4$, as desired. \qed

20.3 Proving $\Psi(G) \leq \sqrt{2\lambda_2}$

In this section we present a proof of the upper bound, $\Psi(G) \leq \sqrt{2\lambda_2}$, based on the proof in [Chu97]. The proof also entails an algorithm due to [Fie73] producing a cut with conductance at most $\sqrt{2\lambda_2}$. Besides the surprising connection to the eigenvalues
of $M$, the algorithm is simple and practical. Based on previous discussions, the reader might be able to guess it.

Recall that

$$\lambda_2 = \min_{y: \langle \sqrt{d}, y \rangle = 0} \frac{\langle y, My \rangle}{\langle y, y \rangle} = \min_{x: \langle d, x \rangle = 0} \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle}.$$ 

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 loosing 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 in fig. 20.2.

**Analysis.** We first prove Cheeger’s inequality, and extract an algorithm from the proof at the end. Call a set $S \subseteq V$ small if $\text{vol}(S) \leq \text{vol}(\overline{S})$. That is, $\Psi(S) = w(\delta(S))/\text{vol}(S)$ for small $S$. The following lemma is the most important part of the proof.

**Lemma 20.5.** Let $x \in \mathbb{R}^V_{\geq 0}$ be nonnegative and $S = \text{support}(x)$. If $S$ is small, then

$$\langle x, Lx \rangle \geq \frac{\Psi^2}{2} \langle x, Dx \rangle.$$ 

**Proof.** Number $S = \{v_1, \ldots, v_k\}$ in decreasing order of $x(v_i)$. For each $i$, let $S_i = \{v_1, \ldots, v_i\}$. Note that each $\text{vol}(S_i)$ is small. We have

$$\Psi(x, Dx) = \Psi \sum_{i=1}^{k} x^2(v_i) \deg(v_i) = \Psi \sum_{i=1}^{k} x^2(v_i)(\text{vol}(S_i) - \text{vol}(S_{i-1}))$$ 

$$\equiv \Psi \sum_{i=1}^{k-1} \text{vol}(S_i) \left( x^2(v_i) - x^2(v_{i+1}) \right) \leq \sum_{i=1}^{k-1} \left( x^2(v_i) - x(v_{i+1}) \right)^2 w(\delta(S_i))$$ 

$$= \sum_{e \in uv} w(e) \left| x^2(u) - x^2(v) \right|$$ 

$$\leq \sqrt{\langle x, Lx \rangle \sum_{e \in uv} (x(u) + x(v))^2}$$ 

$$\leq 2\langle x, Lx \rangle \langle x, Dx \rangle.$$ 

285
20. Conductance

20.3. Proving $\Psi(G) \leq \sqrt{2\lambda_2}$

Kent Quanrud
Fall 2022

<table>
<thead>
<tr>
<th>Fiedler((G = (V, E), \ w \in \mathbb{R}^E_{\geq 0}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let (x) minimize (\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle}) s.t. (\langle x, d \rangle = 0) and (x \neq 0).</td>
</tr>
<tr>
<td>2. Number the vertices (v_1, \ldots, v_n) in order of (x(v_i)). (S_i = {v_1, \ldots, v_i}) for all (i). Return the set (S_i) of minimum conductance.</td>
</tr>
</tbody>
</table>

Figure 20.2: Fiedler’s algorithm for low-conductance cuts.

(a) interchanges sums (where we dropped the term for \(\text{vol}(S_0) = 0\)). (b) is by definition of \(\Psi\). Here we note that if \(\text{vol}(S_i) > \text{vol}(\bar{S}_i)\), then \(x^2(v_i) - x(v_{i+1}^2)\) (c) is by Cauchy-Schwartz. (d) is by the inequality \((a + b)^2 \leq 2(a^2 + b^2)\).

Recall that the algorithm first computes \(x\) such that \(\langle x, Lx \rangle = \lambda_2 \langle x, Dx \rangle\). If \(x\) was nonnegative with small support, then applying lemma 20.5 to \(x\) gives \(\lambda_2 \geq \Psi^2 / 2\), as desired. Of course \(x\) is neither nonnegative nor supported by a small set of vertices. We first address the issue of nonnegativity.

**Lemma 20.6.** Let \(x \in \mathbb{R}^V\), and split \(x = x_+ - x_-\) where \(x_+, x_- \in \mathbb{R}_{\geq 0}^V\) are nonnegative vectors whose nonzeroes are the positive and the (absolute values of the) negative coordinates of \(x\), respectively. Then \(\langle x, Lx \rangle \geq \langle x_+, Lx_+ \rangle + \langle x_-, Lx_- \rangle\) and \(\langle x, Dx \rangle \leq \langle x_+, Dx_+ \rangle + \langle x_-, Dx_- \rangle\).

**Proof.** For the first inequality, we have

\[
\langle x, Lx \rangle = \sum_{e=\{u,v\}} w(e)(x(u) - x(v))^2 = \sum_{e=\{u,v\}} w(e)(x_+(u) - x_+(v) - (x_-(u) - x_-(v)))^2 = \langle x_+, Lx_+ \rangle + \langle x_-, Lx_- \rangle - 2 \sum_{e=\{u,v\}} w(e)(x_+(u) - x_+(v))(x_-(u) - x_-(v)) 
\]

For the last inequality, observe that \((a_+ - b_+)(a_- - b_-) \leq 0\) for any real numbers \(a\) and \(b\). The second inequality is simpler and the proof is left to the reader.

Now we can split \(x\) into two nonnegative vectors \(x_+\) and \(x_-\). However \(x_+\) and \(x_-\) may not have small support. This is addressed by the following lemma that allows us to translate \(x\) before splitting into the positive and negative parts.

---

\(^2\)WLOG \(a \geq b\). Then \(a_+ - b_+ \geq 0\) and \(a_- - b_- \leq 0\).
**Lemma 20.7.** Let \( \langle x, d \rangle = 0 \). For all \( \alpha \in \mathbb{R} \),

\[
\frac{\langle x + \alpha \mathbb{1}, Lx + \alpha \mathbb{1} \rangle}{\langle x + \alpha \mathbb{1}, Dx + \alpha \mathbb{1} \rangle} \leq \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle}.
\]

**Proof.** We have \( \langle x, Lx \rangle = \langle x + \alpha \mathbb{1}, L(x + \alpha \mathbb{1}) \rangle \) because \( \mathbb{1} \in \ker(L) \). Meanwhile, consider the function

\[
f(\alpha) = \langle x + \alpha \mathbb{1}, D(x + \alpha \mathbb{1}) \rangle \overset{(a)}{=} \langle x, Dx \rangle + \alpha^2 \langle d, \mathbb{1} \rangle,
\]

where for (a) we recall that \( \langle d, x \rangle = 0 \). Of course the RHS is minimized at \( \alpha = 0 \). \( \square \)

To complete the proof, let \( x \in \mathbb{R}^V \) with \( \langle x, d \rangle = 0 \) and \( \langle x, Lx \rangle = \lambda_2 \langle x, Dx \rangle \). Then

\[
\langle x, Lx \rangle \geq \langle x_+, Lx_+ \rangle + \langle x_-, Lx_- \rangle \geq \frac{\Psi^2}{2}((\langle x_+, Dx_+ \rangle + \langle x_-, Dx_- \rangle) \geq \frac{\Psi^2}{2} \langle x, Dx \rangle.
\]

Rearranging gives

\[
\lambda_2 = \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} \geq \frac{\Psi^2}{2},
\]

as desired.

**Making the proof algorithmic.** Let \( \alpha \) denote the minimum conductance of all the cuts considered by the algorithm. We can easily adjust the key lemma, lemma 20.5, to incorporate \( \alpha \), as follows.

**Lemma 20.8.** Let \( x \in \mathbb{R}^V_{\geq 0} \) be nonnegative with small support \( S = \text{support}(x) \). Let \( v_1, \ldots, v_k \) be number \( S \) in nonincreasing order of \( x(v_i) \), and let \( S_i = \{ v_1, \ldots, v_i \} \) for all \( i \). If all \( S_i \) have conductance at least \( \alpha \), then \( \langle x, Lx \rangle \geq \alpha^2 \langle x, Dx \rangle / 2 \).

Repeating the same proof as before, except with lemma 20.8 in place of lemma 20.5, gives \( \lambda_2 \geq \alpha^2 / 2 \), hence \( \alpha \leq \sqrt{2\lambda_2} \).

### 20.4 Additional notes and materials

We refer the reader to [Spi19; Tre16] for more on spectral graph theory.

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

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

Exercise 20.1. 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 $x \notin \text{span}(1)$, we have $\langle x, Lx \rangle > 0$.

Exercise 20.2. Finish the proof of lemma 19.3, 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 20.3. Here we prove the lower bound in Cheeger’s inequality (theorem 20.3), $\Psi(G) \geq \lambda_2/2$. Let $S \subseteq V$ induced the minimum conductance cut; i.e., $\text{vol}(S) \leq \text{vol}(V)/2$ and $\Psi(G) = \Psi(S)$. Consider the vector $x = D^{1/2} 1_S$ and let $y = D^{1/2} 1_{\bar{S}}$.

1. Show that
   \[ \frac{\langle x, Mx \rangle}{\langle x, x \rangle} = \frac{\langle 1_S, L1_S \rangle}{\langle 1_S, D1_S \rangle}, \quad \frac{\langle y, My \rangle}{\langle y, y \rangle} = \frac{\langle 1_{\bar{S}}, L1_{\bar{S}} \rangle}{\langle 1_{\bar{S}}, D1_{\bar{S}} \rangle}, \quad \text{and} \quad \langle x, y \rangle = 0. \]

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 21

Deterministic log-space connectivity

21.1 Introduction

Consider the \((s,t)\)-connectivity problems in undirected graphs. Let \(G = (V, E)\) be an undirected graph with \(m\) edges and \(n\) vertices. Let \(s, t \in V\). We want to know if \(s\) and \(t\) are connected in \(G\). This is very easy in normal settings but it is trickier to do in logarithmic space since we cannot mark the vertices as we visit them. Previously we saw that how to solve this with random walks. The algorithm takes a random walk from \(s\) and answers yes if we reach \(t\) within the first \(O(mn)\) steps. The algorithm was justified by analyzing the cover time: we showed that a random walk from \(s\) visits every vertex (connected to \(s\)) in \(\leq 2mn\) steps in expectation. A follow up 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 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. The spectral gap \(\gamma\) is the difference between the maximum eigenvalue 1 and the absolute value of any other eigenvalue of the random walk matrix.

Let \(d \in \mathbb{N}^V\) be the degrees of \(G\), and recall that the stationary distribution (for undirected graphs) is proportional to \(d/2m\). Let \(x_k \in \Delta^V\) denote the distribution after \(k\) random steps (from some arbitrary initial distribution \(x_0 \in \Delta^V\)). Then \(x_k\) converged to the stationary distribution \(s\) at a rate of

\[\|x_k - d/2m\|_2 \leq (1 - \gamma)^k n.\]

An undirected graph is called an expander graph if the spectral gap \(\gamma\) is at least a constant. Suppose \(G\) is an expander with spectral gap (say) 1/2. Then a random
walk converges exponentially fast at a rate of

$$\|x_k - d/2m\|_2 \leq \frac{n}{2^k}.$$ 

For $k = O(\log n)$ steps, we have

$$\|x_k - d/2m\|_\infty \leq \|x_k - d/2m\|_2 \leq \frac{n}{2^k}.$$ 

Since $d(v)/2m \geq 1/2m$ for all $v$, we have $x_k(v) > 0$ for $k \geq O(\log n)$.

This holds for any initial distribution, so suppose we started a single vertex $s$. Then $x_k(v) > 0$ implies that there exists a path from $s$ to $v$ of length $k$. That is, any graph with constant spectral gap has $O(\log n)$ diameter.

Now suppose that $G$ also had maximum degree at most (say) 8. A random walk on $G$ only needs to sample $O(1)$ random bits to sample each random step. If there is a path from $s$ to $t$ of length $O(\log n)$, then there is some sample of $O(\log n)$ random bits that generates a random walk from $s$ to $t$. The number of random bits is so little that we can deterministically enumerate and try all $2^{O(\log n)} = \text{poly}(n)$ possible bit strings!

**Observation 21.1.** $(s,t)$-connectivity in a constant degree expander can be decided deterministically in $\text{poly}(n)$ time and $O(\log n)$ space.

Our goal is to generalize this argument from expanders to general graphs. We will prove the following theorem due from Reingold [Rei08].

**Theorem 21.2** ([Rei08]). There is a $O(\log n)$ space, polynomial time deterministic algorithm for $(s, t)$-connectivity in undirected graphs with $n$ vertices.

### 21.2 An overview of the deterministic connectivity algorithm

We first give a high level of the algorithm and analysis of theorem 21.2. We will introduce some important technical lemma’s, and show how to complete the proof of theorem 21.2 assuming they hold true. We will prove the lemma’s in subsequent sections.

The approach taken by [Rei08] starts from Observation 21.1, that deterministic log-space connectivity is easy on expander graphs. The high-level idea is to implicitly convert the input graph $G$ into a constant degree expander. [Rei08], building on previous work [RVW00], applies a sequence of graph transformations to $G$ that iteratively improves the spectral gap. These transformations are done implicitly
because in $O(\log n)$ space we cannot explicitly rebuild the graph. One can then derandomize the random walk algorithm by enumeration on this artificial expander graph.

We assume the graph $G$ is a regular graph where all vertices have degree $d$. It is not difficult to take the input graph and transform it into a regular graph; we leave this as an exercise to the reader. The graph transformations introduced below preserve the regularity of the graph (though $d$ changes). For the rest of this section, we always let $G$ denote an $n$-regular undirected graph with $n$ vertices and degree $d$.

We now introduce the two graph transformations that we will use.

**Graph transformation 1: Powering.**

For $k \in \mathbb{N}$, the $k$th power of $G$, denoted $G^k$, is the multi-graph on $V$ with edges corresponding to all the $k$-step walks for $G$. That is, for $u, v \in V$, the edge $(u, v)$ has multiplicity equal to the number of walks from $u$ to $v$.

If $G$ has random walk matrix $R$, then $G^k$ has random walk matrix $R^k$, and the eigenvalues of $R^k$ raise the eigenvalues of $R$ to the $k$th power. This improves the spectral gap which is good. Unfortunately powering also increases the maximum degree.

**Lemma 21.3.** $G^k$ is a regular undirected graph on $V$ with degree $d^k$ and random walk map $R^k$. If $R$ has spectral gap $\gamma$, then $R^k$ has spectral gap $1 - (1 - \gamma)^k$.

The proof is left to the reader.

**Graph transformation 2: the zig-zag product.** The second operation is called the zig-zag product. The goal of the zig-zag product is to reduce the degree $d$. The construction is much more subtle than graph powering.

Let $H = (V_0, E_0)$ be a constant degree expander with $d$ vertices and degree $d_0$.

Note that the number of vertices in $H$ matches the degree of $G$ exactly. We identify the vertice of $H$ with the set of indices $[d] = \{1, \ldots, d\}$.

We will use $H$ to simulate the choices of a random walk in $G$. At a high level, we keep one vertex $v_H$ in $H$ and one vertex $v_G$ in $G$. Rather than taking a random step in $G$ (which requires $\log(d)$ bits), we take a random step on $H$ (which requires only $\log(d_0)$ bits), and use the new index of $v_H$ in $H$ to choose a random neighbor of $v_G$ in $G$, and move from $v_G$ to this neighbor. Intuitively, if $H$ is an expander, then the random step in $H$ should behave similarly to sampling a uniformly random
vertex in $H$. In that case the choice of neighbor in $G$ behaves similarly to a uniformly random choice of neighbor, and overall, $v_i$ follows a path similar to a random walk. Meanwhile we only need $\log(d_0)$ bits to implement this "pseudo-random" step.

Now the action just described is not symmetric. More precisely, the arcs these steps describe on the product set $V_1 \times V_2$ do not give an undirected graph. To ensure we have an undirected graph we will have to append a few additional steps.

We are now prepared to define the zig-zag product formally. The zig-zag product, denoted $Z(G \mid H)$, is a regular graph with vertex set $V_1 \times V_2$ and degree $d_0^2$. 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).

To define this formally, let $(v_1, i_1) \in V_1 \times V_2$ be a vertex. For $(k_1, k_2) \in [d_0] \times [d_0]$, the $(k_1, k_2)$th neighbor of $(v_1, i_1)$ is the point $(v_2, i_4)$ obtained by the following steps.

\[
\begin{align*}
(v_1)_{i_1} &\rightarrow (v_1)_{i_2} \\
&\rightarrow (v_2)_{i_2} \\
&\rightarrow (v_2)_{i_3} \\
&\rightarrow (v_2)_{i_4}
\end{align*}
\]

where:

- (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) moves from $i_2$ to $i_3$ where $v_1$ is the $i_3$th neighbor of $v_2$.
- (d) moves $i_3$ to its $k_2$th neighbor in $H$.

The most important steps are the first two. This is where a random step in $H$ induces a step in $G$. The remaining two steps makes the process reversible and the resulting graph an undirected graph. (Verifying this is left to the reader.)

Consider an edge starting from $(v_1, i_1)$ and ending at $(v_2, i_4)$. $v_1$ and $v_2$ are neighbors in $G$. However, $i_1$ and $i_4$ are not (necessarily) neighbors! The discontinuity arises in the third step, where the relationship between $v_1$ and $v_2$ in $G$ is used to "teleport" $i_2$ to $i_3$. Luckily, all we will really care about is preserving the connectivity in $G$. If $s, t \in V$ are connected in $G$, then for each index $i \in [d]$, there is an index $j \in [d]$ such that $(s,i)$ and $(t,j)$ are connected. Conversely, if $(s,i)$ and $(t,j)$ are connected in $Z(G \mid H)$, then $s$ and $t$ are connected in $G$.

Analyzing the zig-zag product is the most involved part of the overall proof. We summarize its properties in the following lemma and defer the proof to later.

**Lemma 21.4.** 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 $\gamma_G \gamma_H^2$. 

292
Combing powering with the zig-zag product. When we combine the zig-zag product with powering we get the following. For technical reasons it is convenient to assume $G$ is regular with degree $d^2$ for some integer $d$.

**Lemma 21.5.** 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 $d^4 n$ 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 $\geq 1/\text{poly}(n)$. We can also assume that there exists a regular expander $H$ on $d^4$ vertices, degree $d$, and spectral gap $\geq 3/4$. If $\gamma_G \leq 1/16$, then by lemma 21.5 $Z(G^2 \mid H)$ has spectral gap

$$\geq 1.01 \gamma_G.$$

That is, we increase the spectral gap by a constant factor! By repeating the construction $O(\log n)$ times, the final graph has constant expansion! The degree, meanwhile, is still $d^2$ — the same as before, and a constant.

We need to check that the number of vertices did not blow up too much. Each time we apply lemma 21.5, the number of vertices increases by a constant factor $d^4$. Therefore, after $O(\log n)$ iterations the number of vertices increases by a poly($n$)-factor. Altogether we have a constant degree graph with poly($n$) vertices and constant spectral-gap — primed for derandomizing the random walk approach.

To keep the space at $O(\log n)$ we do not explicitly construct the generated expander. Instead we will simulate walks on the expander implicitly.

All that said, we still need to verify that we can run simulate a random walk on the generated expander in $O(\log n)$ space. Let $G_0$ denote the input graph (with $n$ vertices and constant degree $d^2$) and let $G_k = Z(G^2_{k-1} \mid H)$ be the graph obtained after the $k$th iteration of lemma 21.5. To simulate a random walk on one of these generated graphs, we need to be able to answer the following $i$th-neighbor query. This query takes as input a vertex $v$ and an index $i$, and returns the $i$th neighbor of $v$. We need to show how to implement each query with small space.

We claim the following for each index $j$.

1. The space required to execute an $i$th-neighbor query on $G^2_j$ is $O(1)$ plus the space required to simulate a step on $G_j$.

2. The space required to execute an $i$th-neighbor query on $Z(G^2_j \mid H)$ is $O(1)$ plus the space required to simulate a step on $G^2_j$. 

293
If the above hold, then the space required to simulate a step on $G_k$ is $O(k)$, as desired.

Consider the first claim, for $G^2_j$. We are given a vertex $v_1$ in $G^2_j$ and two indices $i_1, i_2 \in [d^2]$. We query $(v, i_1)$ to take a step in $G_j$, which returns a vertex $v_2$ in $G_j$. We then query $(v, i_2)$ to take a step in $G_j$ which returns a vertex $w_3$. The maximum amount of space we ever use is $O(1)$ plus the space recursively required to take a step in $G_j$.

Consider the second claim, where we are simulating a step on $Z(G^2_j \mid H)$. We are given a vertex $(v_1, i_1)$, where $v_1$ is a vertex in $G^2_j$ and $i_1$ is a vertex in $H$ (and at most a constant). We are also given two indices $j_1, j_2 \in [d_0]$ and want to return the $(j_1, j_2)$th neighbor of $(v_1, i_1)$. We first take a step in $H$ from $i_1$ to $i_2$, using $O(1)$ space. We then query $G^2_j$ for the $i_2$th edge from $v_1$ to some $v_2$ in $G^2_j$. We then query, for each $i_3 \in [d]$, the $i_3$th edge from $v_2$ in $G^2_j$ 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 $G^2_j$. 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 $G^2_j$.

All put together, for $k = O((log(n))$, $G_k$ is a constant degree expander with poly($n$) that preserves connectivity from $G$. Moreover, we can navigate $G_k$ implicitly via $i$th-neighbor queries in $O((log n))$ space per query. We derandomize the random walk algorithm on $G_k$, which gives a deterministic algorithm for connectivity in $G$.

Up to proving lemma 22.8 in the subsequent section, this completes the proof of theorem 21.2.

21.3 Preliminaries

We introduce some mathematical background needed to analyze the zig-zag product.

21.3.1 Tensors of linear maps

Recall that $\mathbb{R}^{m \times n}$ denotes the space of linear maps from the vector space $\mathbb{R}^n$ to the vector space $\mathbb{R}^m$. $\mathbb{R}^{m \times n}$ itself is a real vector space with inner product given by

$$\langle A, B \rangle = \sum_{i,j} A_{ij} B_{ij} = \text{trace}(A^T B).$$

Two helpful identities are

$$\langle A, c \otimes d \rangle = \langle c, Ad \rangle$$

$$\langle (a \otimes b), (c \otimes d) \rangle = \langle a, c \rangle \langle b, d \rangle$$

294
21. Deterministic log-space connectivity

21.3. Preliminaries

for $A \in \mathbb{R}^{m \times n}$, $a, c \in \mathbb{R}^m$ and $b, d \in \mathbb{R}^n$, which the reader may verify.

For $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{n \times n}$, the outer product (a.k.a. the tensor product) of $A$ and $B$, denoted $A \otimes B$, is the linear map defined by

$$\langle x, ((A \otimes B)C)y \rangle = \langle A^T x, CB^T y \rangle,$$

or equivalently,

$$(A \otimes B)C \overset{\text{def}}{=} ACB^T,$$

for $C \in \mathbb{R}^{m \times n}$. We leave it to the reader to verify that this defines a linear map over $\mathbb{R}^{m \times n}$. We have the identities

$$(A \otimes B)(c \otimes d) = Ac \otimes Bd,$$  \hspace{1cm} (21.1)

$$(A \otimes (B + C)) = (A \otimes B) + (B \otimes C),$$  \hspace{1cm} (21.2)

$$(A \otimes B)^T = (A^T \otimes B^T)$$  \hspace{1cm} (21.3)

which are also left to the reader to verify. The last identify implies that if $A$ and $B$ are symmetric, then so is $A \otimes B$.

Lemma 21.6. $(A \otimes B)$ has eigenvalue-eigenvectors pairs of the form $(\lambda_A \lambda_B, x \otimes y)$, where $x$ is an eigenvector of $A$ with eigenvalue $\lambda_A$ and $y$ is an eigenvector of $B$ with eigenvalue $\lambda_B$.

Proof. For each pair $(\lambda_A, x)$ and $(\lambda_B, y)$, we have

$$(A \otimes B)(x \otimes y) = Ax \otimes By = \lambda_A x \otimes \lambda_B y = \lambda_A \lambda_B (x \otimes y),$$

so $(x \otimes y)$ is an eigenvector of $(A \otimes B)$ with eigenvalue $\lambda_A \lambda_B$. There are $m$ choices of $(\lambda_A, x)$ and $n$ choices of $(\lambda_B, y)$ so together this gives $mn$ eigenvalue/vector pairs. Since $(A \otimes B)$ acts on an $(m \times n)$-dimensional vector space, these are all the eigenvalue/vector pairs.

Tensor products of random walks. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs with random walk matrices $R_1$ and $R_2$. Consider the random walk on $V_1 \times V_2$ where given a pair of vertices $(v_1, v_2) \in V_1 \times V_2$, we take a random step from $v_1$ to $w_1$ according to $R_1$ and a random step from $v_2$ to $w_2$ according to $R_2$. The distribution of $(w_1, w_2)$ is described by the $m \times n$ matrix

$$(R_1 1_{v_1}) \otimes (R_2 1_{v_2})$$

where $1_v$ denotes the \{0, 1\}-indicator vector for a vertex $v$. More generally, given a distribution $P \in \mathbb{R}_{\geq 0}^{m \times n}$ over $V_1 \times V_2$, taking random steps along $R_1$ and $R_2$ simultaneously gives the distribution

$$(R_1 \otimes R_2)P = R_1^T PR_2.$$
21.3.2 The operator norm

The operator norm of a linear map $A$ is defined by
\[ \|A\| = \sup\{\|Ax\| : \|x\| = 1\}. \]

$\|A\|^2$ is the maximum eigenvalue of the positive semi-definite matrix $A^T A$. For a symmetric map $A$, this is the maximum absolute eigenvalue of $A$.

We have the identities
\[
\begin{align*}
\|A + B\| &\leq \|A\| + \|B\|, \\
\|AB\| &\leq \|A\|\|B\|, \\
\|A \otimes B\| &\leq \|A\|\|B\|,
\end{align*}
\]
the proofs of which are left to the reader.

21.4 Analysis of the zig-zag product

Lemma 22.8. 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 $Z(G \mid H)$ is a regular undirected graph with $nd$ vertices, degree $d_0^2$ and spectral gap $\gamma_G \gamma_H^2$.

Let $R_Z$ denote the random walk matrix of $Z(G^2 \mid H)$. Let $R_H$ be the random walk matrix of $H$. We can write a random step in the zig-zag graph as
\[ R_Z = (I \otimes R_H)Z(I \otimes R_H), \]
where $(I \otimes R_H)$ represents the action where we take a single random step in $H$ but leave the $G$-coordinate fixed, and $Z$ is the (deterministic) zig-zag step that updates the $G$-coordinate and transports the $H$-coordinate, as described above. Ultimately, we want to analyze the spectral gap of $(I \otimes R_H)Z(I \otimes R_H)$.

To give some intuition, recall that $H$ is an expander. That is, taking a few steps in $R_H$ is almost as random as sampling a uniformly random vertex from $H$. To formalize this connection, $S = (1 \otimes 1)/d : \Delta_H \rightarrow \Delta_H$ be the “random walk” that samples a vertex from $H$ uniformly at random. Intuitively, $R_H \approx S$. Suppose we substitute $S$ for $R_H$ in our expression for the random walk in the zig-zag product, giving,
\[ (I \otimes S)Z(I \otimes S). \]
This step describes a zig-zag product of $G$ with the graph $H'$, which is a complete graph with a self-loop at every vertex. Let us walk through a random step in the zig-zag product $Z(G \mid H')$. 

296
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 of \(Z(G | H')\) is precisely \(\gamma_G\), the spectral gap of \(G\).

Of course, \(S\) is not exactly \(R_H\), and additionally, the zig-zag product of \(G\) with \(H'\) does not decrease the degree as we would like. (In fact, it increases the degree.) It remains to quantify the difference between \((I \otimes R_H)Z(I \otimes R_H)\) and \((I \otimes S)Z(I \otimes S)\), which reflects the difference between \(R_H\) and \(S\). As we will make more explicit below, the spectral gap of \(H\), \(\gamma_H\), is also a reflection of the difference between \(R_H\) and \(S\). (\(S\) has spectral gap 1) This difference between \(R_H\) and \(S\), and the correspondance between the difference and \(\gamma_H\), is why the spectral gap decreases from \(\gamma_G\) to \(\gamma_G\gamma_H^2\).

Previously when analyzing the spectral gap of undirected random walks, we applied the spectral theorem via similarity to the normalized random walk matrix. But if the graph is regular, then the random walk matrix is already symmetric, and in fact the same as the normalized random walk matrix. This applies to \(G\), \(H\), and \(Z(G | H)\).

\(Z(G | H)\) has stationary distribution \((\mathbb{1} \otimes \mathbb{1})/d^2\). Therefore \(\mathbb{1} \otimes \mathbb{1}\) is the first eigenvector of \(R_Z\). To bound the spectral gap of \(R_Z\), we want to bound

\[
|\langle x, R_Z x \rangle|
\]

over all \(x\) orthogonal to \(\mathbb{1} \otimes \mathbb{1}\).

We can apply the spectral theorem to \(R_H\) by similarity to the normalized random walk matrix. Here, because \(H\) is a regular graph, \(R_H = A/d\) is already a symmetric matrix (and coincides with the normalized random walk matrix) and we can apply the spectral theorem directly. By the spectral theorem for symmetric maps, combined the the Perron-Frobenius theorem for random walks, we have

\[
R_H = u_1 \otimes u_1 + \lambda_2 (u_2 \otimes u_2) + \cdots + \lambda_n (u_n \otimes u_n),
\]
where \( u_1, \ldots, u_n \in \mathbb{R}^d \) 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
\[
R_H = \frac{1}{d}(\mathbb{1} \otimes \mathbb{1}) + \lambda_2(u_2 \otimes u_2) + \cdots + \lambda_n(u_n \otimes u_n)
= S + \lambda_2(u_2 \otimes u_2) + \cdots + \lambda_n(u_n \otimes u_n).
\]

Let \( R_H' = R_H - \gamma_H S \); then \( R_H' \) has all its eigenvalues in the range \([\gamma_H - 1, 1 - \gamma_H]\).

We factor \( R_Z \) as
\[
(I \otimes R_H)Z(I \otimes R_H) = (I \otimes (\gamma_H S + R_H'))Z(I \otimes (\gamma_H S + R_H'))
= (\gamma_H(I \otimes S) + (I \otimes R_H'))Z(\gamma_H(I \otimes S) + (I \otimes R_H'))
= \gamma_H^2(Z(I \otimes S) + \gamma_H(I \otimes S)Z(I \otimes R_H')
+ \gamma_H(I \otimes R_H')Z(I \otimes S) + (I \otimes R_H')Z(I \otimes R_H').
\]

Let us first analyze the last 3 terms.

Let \( x \in \mathbb{R}^{V \times d} \) be any unit vector orthogonal to the uniform distribution. We have
\[
|\langle x, (I \otimes S)Z(I \otimes R_H')x \rangle| = \|Z(I \otimes R_H')x\|
\leq \|(I \otimes S)Z(I \otimes R_H')\|
\leq 1 - \gamma_H.
\]

the third term contributes \( 1 - \gamma_H \) (times another \( \gamma_H \)) and the fourth term contributes \((1 - \gamma_H)^2\).

For the finale of our analysis, consider the remaining term, \( (I \otimes S)Z(I \otimes S) \). As observed earlier, we have
\[
(I \otimes S)Z(I \otimes S) = (A \otimes S)
\]

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
\[
|\langle x, (I \otimes S)Z(I \otimes S)x \rangle| = |\langle x, (A \otimes S)x \rangle| \leq 1 - \gamma_G.
\]

Adding everything together we have
\[
\gamma_H^2(1 - \gamma_G) + 2\gamma_H(1 - \gamma_H) + \left(1 - \gamma_H^2\right) = 1 - \gamma_G \gamma_H^2,
\]
as desired.
21.5 Additional notes and materials

This topic is also covered in [HLW06, §9] and [Vad12, §4.4], which discuss additional related topics.

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

21.6 Exercises


Exercise 21.2. Prove that the zig-zag product $Z(G | H)$ is an undirected graph.

Exercise 21.3. Prove that the tensor product $A \otimes B$ defines a linear map over $\mathbb{R}^{m \times n}$ for $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{n \times n(437,191),(691,206)}$. 

Exercise 21.4. Prove the tensor product identities in eqs. (21.1)–(21.3).

Exercise 21.5. Prove the operator norm inequalities in eqs. (21.4)–(21.6).

Exercise 21.6. The goal of this exercise is to develop a randomized construction of a constant degree expander.

Let $d \in \mathbb{N}$ be a parameter to be determined. Let $n$ be even. Consider the random graph over $n$ vertices where the edges are the disjoint union of (the edges of) $d$ uniformly random perfect matchings over the vertices. ($G$ can have parallel edges.) We will first show that $G$ has conductance at least $c$ for some constant $c > 0$. The connection to constant degree expanders is made at the end.

The key claim is as follows.

With nonzero probability, for all disjoint $S$ and $T$ such that $k = |S| \leq n/2$ and $|T| = k/6$, some vertex in $S$ is matched to a vertex outside $S \cup T$.

1. Show that the claim implies that $|\delta(S)| \geq |S|/6$ for all $S$ with $|S| \leq n/2$.

2. Show that if also $d = O(1)$, then the claim above implies that the union of matchings gives a graph with constant conductance.
We will prove this claim probabilistically: our first goal is to fix $S$ and $T$, and bound the probability that all of $S$ is matched to $S \cup T$. To this end, for fixed $S$ and $T$, consider the following randomized procedure which produces a perfect matching.

1. Index the vertices $v_1, \ldots, v_n$ 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 $v_j$ from the remaining unmatched vertices (excluding $v_i$) uniformly at random.
   C. Match $v_i$ with $v_j$.

3. Prove that the procedure above generates a uniformly random perfect matching.
4. Show that the probability that $S$ is matched to $(S \cup T)$ in all $d$ matchings is at most $\binom{n}{k}^{-\alpha d}$. for some constant $\alpha > 0$. (I got $\alpha = 1/6$.)$^1$
5. Now prove the key claim above for some constant $d$.

Having now established that $O(1)$ random matchings have constant conductance with high probability:

6. Design and analyze an algorithm that produces a constant degree expander.
   (That is, a regular constant degree graph whose random walk has constant spectral gap $\gamma$.)$^2$

---

$^1$For a single matching, the probability that $S$ is matched to $(S \cup T)$ is bounded above by the probability that the first $k/2$ choices of $v_j$’s are in $S \cup T$.
$^2$Hint: don’t forget about the minimum eigenvalue.
Chapter 22

Reducing randomness with random walks

22.1 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 extends to randomized algorithms as follows.

Definition 22.1. 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 as a strong indicator for the existence of a deterministic one. Still, we do not know really if \( P \) equals \( \text{RP} \) or if \( \text{RP} \) equals \( \text{BPP} \).
There is theoretical interest, sometimes under the heading of pseudorandomness, in a refined understanding of how much randomness is required for various problems. While the holy grail, \( P \) vs \( BPP \), is hard to attack, there is a rich body of literature and results moving towards a conclusion, producing many algorithmic ideas of independent interest along the way.

Let \( L \in \text{RP} \), 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. This takes \( O(m \log(1/\delta)) \) random bits total. While we typically de-emphasize the logarithmic overhead incurred from repetition, it is a deep and natural question to ask if one can reduce or avoid the logarithmic overhead. Surprisingly, one can:

**Theorem 22.2.** Given an algorithm in \( \text{RP} \) that uses \( m \) random bits and has probability of error at most \( 1/2 \), 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.

There is an analogous result for \( \text{BPP} \). The algorithm is the same as for \( \text{RP} \), while the analysis is different.

**Theorem 22.3.** 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.

The rest of this discussion is about proving theorems 22.2 and 22.3.

### 22.2 High level overview: amplification by random walks

The algorithm that obtains the better-than-repetition amplification above is conceptually very clean. Let \( G \) be a constant degree expander with vertex set \( V = \{0,1\}^m \) — that is, a vertex for every possible bit string of \( m \) bits. (We will have to address how to implicitly build such a \( G \) later, but for now let us assume \( G \) is given.) First, select a uniformly random vertex \( v_0 \in V \). (This takes \( m \) random bits). Then take a random walk in \( G \) for \( O(\log(1/\delta)) \) steps. Each step takes \( O(1) \) random bits. For each vertex \( v_i \) along the walk, use \( v_i \) as the input for a new instance of the algorithm. For algorithms in \( \text{RP} \), output the disjunction (the “or”) of all the outputs. For algorithms in \( \text{BPP} \), output the majority vote.

It is easy to see that we use \( m + O(\log(1/\delta)) \) random bits in total. To complete the proofs of theorems 22.2 and 22.3 there are two basic issues to address.
1. We need to show that the bit strings generated by the expander are (probably) useful, for both the \textbf{RP} and \textbf{BPP} settings.

2. We need to show how to efficiently make such a graph \( G \).

For the second point, since \( G \) is exponential-size, this construction has to be implicit.

**Random walks for \textbf{RP}**. Consider the first point — why bit strings generated by an expander graph act like totally random bit strings — for \textbf{RP}. The key lemma is as follows.

**Lemma 22.4.** 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 22.3. First let us derive theorem 22.2 (on amplifying \textbf{RP} via expanders) in light of lemma 22.4.

Let \( B \) be the set of vertices corresponding to “bad” bit strings causing the randomized algorithm to err. Amplifying the original algorithm by a constant number of repetitions as needed, we can make \( \mu \) arbitrarily small; say, \( 1/2 \). By lemma 22.4, if we take a random walk on an expander of bit string, the probability that the random walk stays in \( B \) — and all bit-strings are bad — drops at a rate of \( (1 - \gamma/2)^t \).

**Random walks for \textbf{BPP}**. We now present a similar lemma that is important for \textbf{BPP}.

**Lemma 22.5.** 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

\[
\mu = \mathbb{E}[f(v)] \text{ where } v \in V \text{ is sampled uniformly at random}.
\]

Consider a random walk \( v_1, v_2, \ldots, v_k \in V \) where \( v_1 \in V \) is chosen uniformly at random. Then for all \( \beta > 0 \),

\[
P \left[ \left| \frac{1}{k} \sum_{i=1}^{k} f(v_i) - \mu \right| \geq \epsilon \mu + \beta \right] \leq c e^{ck((1+\epsilon)\epsilon - \beta)}
\]

303
for a universal constant $c > 0$. In particular, for (say) $\epsilon \leq \mu/4$, we have

$$P \left[ \frac{1}{k} \sum_{i=1}^{k} f(v_i) - \mu \geq \epsilon \mu \right] \leq ce^{-\epsilon k \mu/c}$$

for a universal constant $c > 0$.

Analogous to our discussion for RP, lemma 22.5 implies theorem 22.3. We prove lemma 22.5 in section 22.4.

**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 22.5.

**Lemma 22.6.** Let $d \in \mathbb{N}$, and Let $H$ be a regular undirected graph with $d^8$ vertices, degree $d$, and spectral gap $\geq 7/8$. Define graphs $G_1, G_2, \ldots$ by

$$G_1 = H^2$$

$$G_{t+1} = \mathcal{S} \left( (G_t \otimes G_t)^2 \bigg| H \right)$$

Then $G_t$ has spectral gap $1/2$ and $2^{\Omega(2^t)}$ vertices. Simulating one step of a random walk in $G_t$ takes $2^{O(t)}$ time and $O(\log(d))$ random bits.

Taking $t = \log \log m + O(1)$ gives the desired expander over $\{0, 1\}^m$.

**22.3 Amplifying RP: proof of lemma 22.4**

Let $R$ be the random walk matrix on $G$. Let $P : \mathbb{R}^V \to \mathbb{R}^V$ be the projection onto $\mathbb{R}^B$; that is,

$$(Px)_v = \begin{cases} x_v & \text{if } v \in B \\ 0 & \text{otherwise.} \end{cases}$$

$P$ is a linear function with $P = P^T$ and $P^2 = P$. One can think of $P$ as the identity matrix restricted to $\mathbb{R}^B$ (and 0 everywhere else).

Consider the product $PRP$. Given a nonnegative vector $x$, $PRPx$ drops all the mass outside of $B$, take a step according to $R$, and again drops all of the mass outside of $B$. In particular, the probability that the entire walk stays in $B$ is

$$\left\langle 1, (PRP)^t(1/n) \right\rangle.$$
We claim that $PRP$ has maximum eigenvalue $\leq 1 - (1 - \mu)\gamma$. If so, then

$$\langle 1, (PRP)^t(1/n) \rangle = \frac{\langle 1, (PRP)^t 1 \rangle}{\langle 1, 1 \rangle} \leq (\gamma\mu + 1 - \gamma)^t,$$

which completes the proof.

Since $G$ is regular, $R$ is symmetric, and we can write

$$R = \frac{\gamma}{n} (1 \otimes 1) + R',$$

where $R'$ has all of its eigenvalues in the range $[1 - \gamma, \gamma - 1]$. Then

$$PRP = \frac{\gamma}{n} P(1 \otimes 1)P + PR'P.$$

We leave it to the reader to show that $PR'P$ has all its eigenvalues in the range $[1 - \gamma, \gamma - 1]$.

Consider the first term $(\gamma/n)P(1 \otimes 1)P$. We claim that it has maximum eigenvalue $\gamma\mu = |B|/n$, which would give the overall bound of

$$\|PRP\| = \left\| \frac{\gamma}{n} P(1 \otimes 1)P \right\| + \|PR'P\| \leq \gamma\mu + 1 - \gamma,$$

as desired, where $\|\cdot\|$ is the operator norm.

We have

$$\frac{\gamma}{n} P(1 \otimes 1)P = \frac{\gamma}{n} (P \mathbb{1} \otimes P \mathbb{1}).$$

The maximum eigenvector of the outer product $(P \mathbb{1} \otimes P \mathbb{1})$ is (proportional to) $P \mathbb{1}$, with eigenvalue

$$\frac{\langle P \mathbb{1}, P \mathbb{1} \rangle^2}{\langle P \mathbb{1}, P \mathbb{1} \rangle} = \langle P \mathbb{1}, P \mathbb{1} \rangle = |B|.$$

Thus

$$\left\| \frac{\gamma}{n} (P(1 \otimes 1)P) \right\| = \frac{\gamma|B|}{n} = \mu.$$

This gives the desired bound.
22.4 Efficiently amplifying BPP

**Lemma 22.5.** 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

$$\mu = \mathbb{E}[f(v)]$$

where $v \in V$ is sampled uniformly at random.

Consider a random walk $v_1, v_2, \ldots, v_k \in V$ where $v_1 \in V$ is chosen uniformly at random. Then for all $\beta > 0$,

$$P\left[\left| \frac{1}{k} \sum_{i=1}^{k} f(v_i) - \mu \right| \geq \epsilon \mu + \beta \right] \leq c e^{\epsilon k(1 + \epsilon) \mu - \epsilon \beta}$$

for a universal constant $c > 0$. In particular, for (say) $\epsilon \leq \mu / 4$, we have

$$P\left[\left| \frac{1}{k} \sum_{i=1}^{k} f(v_i) - \mu \right| \geq \epsilon \mu \right] \leq c e^{-\epsilon k \mu}$$

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 union bound over both. We have

$$P\left[\sum_{i=1}^{k} f(v_i) \geq k(1 + \epsilon) \mu + \beta \right] \leq \mathbb{E}\left[e^{\epsilon \sum_{i=1}^{k} f(v_i)}\right] e^{-\epsilon (1 + \epsilon) k \mu - \epsilon \beta}. \quad (22.1)$$

The key identity is

$$\mathbb{E}\left[e^{\epsilon \sum_{i=1}^{k} f(v_i)}\right] = \frac{1}{n} \left\langle 1, F(RF)^{k} 1 \right\rangle$$

where $F = \text{diag}(e^{\epsilon f(v_1)}, \ldots, e^{\epsilon f(v_n)})$ 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 $e^{\epsilon f(v)}$ along the walk. We claim the following.

**Claim 1.** \( \frac{1}{n} \left\langle F^{1/2} 1, F(RF)^{k} F^{1/2} 1 \right\rangle \leq e^{\epsilon (1 + \epsilon) \mu + \epsilon \beta})
Assuming claim 1 holds, we have

\begin{equation}
(22.1) \leq e\epsilon \cdot e^{d((1+\epsilon)\epsilon - \beta)},
\end{equation}
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

\[ F^{1/2}R F^{1/2} = \frac{1 - \epsilon}{n} F^{1/2}(\mathbb{1} \otimes \mathbb{1}) F^{1/2} + F^{1/2} R' F^{1/2}. \]

We claim the following.

**Claim 2.** \( F^{1/2}R' F^{1/2} \) has its eigenvalues between \([\epsilon e^\epsilon, -\epsilon e^\epsilon]\).

**Claim 3.** \( F^{1/2}(\mathbb{1} \otimes \mathbb{1}) F^{1/2} \) has maximum eigenvalue \( E\left[e^{\epsilon f(v)}\right]. \)

For the first claim regarding \( F^{1/2}R' F^{1/2} \), for any vector \( x \)

\[
\langle x, F^{1/2}R' F^{1/2} x \rangle = \langle F^{1/2}x, R'(F^{1/2}x) \rangle \leq \epsilon \| F^{1/2}x \|^2 \\
= \epsilon \langle x, Fx \rangle \leq \epsilon e^\epsilon \| x \|^2,
\]

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

\[
\| F^{1/2} \mathbb{1} \|^2 = \langle \mathbb{1}, F \mathbb{1} \rangle = \sum_v e^{\epsilon f(v)} = n \mathbb{E}\left[e^{\epsilon f(v)}\right].
\]

This establishes the second claim.

Combining the two claims above, we have that \( F^{1/2}R F^{1/2} \) has maximum (absolute) eigenvalue

\[
\| F^{1/2} R F^{1/2} \| \leq \epsilon e^\epsilon + (1 - \epsilon) \mathbb{E}\left[e^{\epsilon f(v)}\right].
\]
We expand the right hand side by the inequality $\epsilon^2 \leq 1 + \epsilon + \epsilon^2$ for small $|t|$, giving the upper bound

$$\|F^{1/2}RF^{1/2}\| \leq \epsilon(1 + \epsilon + \epsilon^2) + (1 - \epsilon)\mathbb{E}[1 + \epsilon f(v) + \epsilon^2 f(v)]$$

$$= \epsilon(1 + \epsilon + \epsilon^2) + (1 - \epsilon)(1 + \epsilon \mu + \epsilon^2 \mu)$$

$$\leq 1 + (\epsilon + \epsilon^2)(\mu + \epsilon)$$

$$\leq e^{(\epsilon + \epsilon^2)(\mu + \epsilon)}$$

In turn, for the $k$th power, we have

$$\|\left(F^{1/2}RF^{1/2}\right)^k\| = \|F^{1/2}RF^{1/2}\|^k \leq e^{k(1+\epsilon)(\mu+\epsilon)}.$$ 

Finally, returning to the original quantity we wanted to sum, we have

$$\frac{1}{n}\left\langle F^{1/2}1, F(RF)^kF^{1/2}1 \right\rangle \leq \epsilon^{k(1+t)(\mu+\epsilon)}\mathbb{E}e^{\epsilon f(v)} \leq (1 + O(\epsilon))e^{(k(1+\epsilon)(\mu+\epsilon))}.$$ 

as desired for Claim 1. 

22.5 Efficiently making large expanders

It remains to be shown that large expanders can be constructed efficiently. Previously, in the interest of deterministic connectivity, we studied the amplification

$$G \mapsto \mathcal{Z}(G^2 \mid H),$$

where the degrees and sizes of $G$ and $H$ are appropriately set. The primary goal of the goal of that exercise was to increase the spectral gap given an input graph (with bad spectral gap) in a space efficient manner.

Here our goal is slightly different, because simply want to make an expander over $2^m$ vertices without the burden of some bad input graph. That is, we simply want to make a large - very, very large - expander. Note that we want to make this graph implicitly and be able to take a step in the graph in $O(\text{polylog}(m))$ time per step – importantly, this is doubly logarithmic in the number of vertices, $2^m$. If we apply the construction form connectivity starting from a constant sized expander, we will end up needing $O(m)$ iterations to get up to $2^m$ vertices, since each iteration increases the number of vertices of a constant factor. Thus, in contrast to before, the goal is to increase the number of vertices given an expander as efficiently as possible.

Let us now restate the main lemma that we need to prove.
Lemma 22.6. Let $d \in \mathbb{N}$, and let $H$ be a regular undirected graph with $d^8$ vertices, degree $d$, and spectral gap $\geq 7/8$. Define graphs $G_1, G_2, \ldots$ by

$$G_1 = H^2$$
$$G_{t+1} = Z((G_t \otimes G_t)^2 \mid H)$$

Then $G_t$ has spectral gap $1/2$ and $2^{\Omega(t^2)}$ vertices. Simulating one step of a random walk in $G_t$ takes $2^{O(t^3)}$ time and $O(\log(d))$ random bits.

We first recall the first two lemma’s that we proved previously.

Lemma 22.7. 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 22.8. 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 $Z(G \mid H)$ is a regular undirected graph with $nd$ vertices, degree $d_0^2$ and spectral gap $\gamma_G \gamma_H^2$.

The second lemma, regarding the zig-zag product, required the following structural lemma about the tensor product of undirected graphs and their random walks.

Lemma 22.9. 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 $\lambda_1\lambda_2$.

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

$$G_{i+1} = Z((G_i \otimes G_i)^2 \mid H).$$
The various parameters of interest develop as follows.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$G \rightarrow G \otimes G \rightarrow (G \otimes G)^2 \rightarrow Z((G \otimes G)^2 \mid H)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>$n \rightarrow n^2 \rightarrow n^2 \rightarrow n^2 d^4$</td>
</tr>
<tr>
<td>Degree</td>
<td>$d^2 \rightarrow d^4 \rightarrow d^8 \rightarrow d^2$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$\gamma \rightarrow \gamma \rightarrow 2\gamma - \gamma^2 \rightarrow (2\gamma - \gamma^2)(7/8)^2$</td>
</tr>
</tbody>
</table>

We note that $\gamma \geq 1/2$ implies $(2\gamma - \gamma^2) \geq 1/2$, so the spectral gap never drops below 1/2.

### 22.6 Additional notes and materials

See [Vad12] for additional topics in pseudorandomness.

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

### 22.7 Exercises

**Exercise 22.1.** The definition of RP allows for error probability 1/2 while BPP requires a constantly strictly less than 1/2 (e.g., 1/3). Here we explore why.

Give a concise description of the family of all languages $L$ for which there is a randomized polynomial time algorithm that, given input $x$:
1. If $x \in L$, decides that $x \in L$ with probability at least 1/2.
2. If $x \notin L$, decides that $x \notin L$ with probability at least 1/2.

**Exercise 22.2.** Extend lemma 22.4 to graphs $G$ that are not regular.

**Exercise 22.3.** Extend lemma 22.5 to graphs $G$ that are not regular.
Chapter 23

Randomized Proofs and Verification by Random Walks

23.1 Randomized Proofs

Recall that the class $P$ is the class of all polynomial time solvable problems, and $NP$ is the class of all languages that can be decided in non-deterministic polynomial time. Equivalently, a language $L$ is in $NP$ if a membership $x \in L$ can be proven in polynomial time. This means there exists a (deterministic) polynomial time algorithm, called the verifier. The verifier takes as input $x \in \{0, 1\}^n$ and an additional polynomial-sized input $y \in \{0, 1\}^{\text{poly}(n)}$ called the proof. Based on $x$ and $y$, the verifier decides if $x \in L$ according to the following protocol:

- If $x \in L$, then the verifier accepts $x$ for some $y$.
- If $x \notin L$, then the verifier rejects $x$ for all $y$.

Obviously $P \subseteq NP$, and an outstanding open problem is whether $P$ is equal to $NP$. A long track record of failing to solve many $NP$ problems of practical interest, combined with the equivalence class of $NP$-complete problems, suggests that they are not equal, but again, we cannot prove it. $P$ vs $NP$ is a fascinating question extending beyond computation; see for example [Wig09].

Consider these questions from a randomized point of view. Suppose we granted the verifier access to randomization, and relaxed our guarantee to have one-sided error. Consider the following protocol for input $x$ and proof $y$:

- If $x \in L$, then for some $y$, the verifier always accepts $x$.
- If $x \notin L$, then for all $y$, the verifier accepts $x$ with probability $\leq 1/2$.

The proof is accessed in an oracle model, where the verifier can query the $i$th bit from an oracle. For a language $L$, a (nonadaptive) probabilistically checkable proof with $r$ random bits and $q$ queries, denoted $PCP(r,q)$ is one that, given an input $x$ and oracle access to a proof $y$, decides if $x \in L$ via the following steps:

1. Given full access to $x$ and $O(r)$ random bits, the verifier chooses $O(q)$ locations in $y$ to query.
23. Randomized Proofs and Verification by Random Walks

23.1. Randomized Proofs

Kent Quanrud
Fall 2022

The verifier makes these $O(q)$ queries.

3. Based on $x$, the $O(r)$ random bits from (1), and the $O(q)$ queries in (2), the verifier decides if $x \in L$.

The verifier can spend polynomial time inspecting the input $x$ and the outcomes of the $O(r)$ coin flips to choose its queries. It is important that the queries to the proof are nonadaptive — the $i$th query does not depend on the outcome of the previous $i-1$ queries.

Of course any language on NP has such a verifier; i.e., $\text{NP} \subseteq \text{PCP}(0, \text{poly}(n))$. The question is, with randomization in hand, can $q$ be reduced from $\text{poly}(n)$ to $n^{o(1)}$?

**Theorem 23.1.** $\text{NP} = \text{PCP}(O(\log n), O(1))$.

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 large impact in hardness of approximation. As we know all too well, many problems are NP-hard, which motivates approximation algorithms for these problems. Are there also limits to approximations? Can we expect better and better approximations over time for all problems, or is there also a hard limit to approximations?

As a concrete example, consider the 3-SAT problem. The input is a SAT formula with exactly 3 literals per clause, and (in the optimization formulation) the goal is to satisfy as many clauses as possible. As discussed previously, if we randomly assign each variable, then we get a $7/8$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 23.2** ([Hås01]). For all $\epsilon > 0$, it is NP-Hard to obtain a $(7/8 + \epsilon)$-approximation to 3-SAT.

There are hardness of approximation results for many problems, connected to one another via approximation-preserving reductions.

One direction of the PCP theorem is easier to prove than the other and left as an exercise to the reader.

**Exercise 23.1.** Prove that $\text{PCP}(O(\log n), O(1)) \subseteq \text{NP}$.

It remains to show $\text{NP} \subseteq \text{PCP}(O(1), O(\log n))$: that is, every language in NP has a probabilistically checkable proof.

\(^{1}\)The old joke is that the PCP theorem implies a much faster way to grade algorithms homework.
The celebrated PCP theorem was developed in the late 80’s and early 90’s and was born out of earlier developments investigating interactive proofs. Unfortunately the mathematical techniques used in this original line of work have not been developed in this class. Instead will present a more recent proof by Dinur [Din07] that is considered to be simpler then the original proof of the PCP theorem. The machinery driving Dinur’s approach will relate to recent discussions on random walks.

23.2 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 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. For a CSP \( P \), we let \( \text{unsat}(P) \) denote the minimum fraction of unsatisfied clauses of \( P \) over all \( P \). (e.g., \( \text{unsat}(P) = 0 \) iff \( P \) is satisfiable.)

Our discussion is about proving the following theorem in particular.

**Theorem 23.3.** There are integers \( q > 1, |A| > 1 \) such that, given a \( q \)-ary CSP over alphabet \( A \), it is NP-hard to decide whether

1. All clauses can be satisfied (\( \text{unsat}(P) = 0 \)).
2. Less than or equal to half the clauses can be satisfied \( \text{unsat}(P) \geq 1/2 \).

It is equivalent to the PCP theorem.

**Theorem 23.4.** Theorems 23.1 and 23.3 are equivalent.

The proof is left to the reader in the following exercise.

**Exercise 23.2.** Prove theorem 23.1 and theorem 23.3 are equivalent. Below we give part of the proofs, in both directions, to get you started.

1. **Theorem 23.1 \( \implies \) theorem 23.3.** Suppose the PCP theorem, theorem 23.1, 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.
23. Randomized Proofs and Verification by Random Walks

23.3 Graph CSP, and amplification

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 23.3 $\implies$ theorem 23.1. 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 probabilisticaly checkable proof system where...

23.3 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 23.3.

Binary CSP can be modeled as a graph problem. We think of each variable $v \in V$ as a vertex. For every clause over two variables $v_1, v_2$, we have an edge between $v_1$ and $v_2$ labeled by that clause. Note that we can have parallel edges if there are multiple clauses for the same pair of vertices. To emphasize this graphical viewpoint, we call binary CSP problems graph CSP from now on.

Graph CSP is NP-Hard even for 3 letters in the alphabet. In particular, 3-colorability\(^2\) is a special case of graph CSP that is NP-Hard. In our CSP terminology, this means it is NP-Hard to decide if $\text{unsat}(G) = 0$. 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 some fixed constant $c$.

\(^2\)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.
Iterative amplification - an overview. For inspiration, we briefly recall the deterministic logspace algorithm for \((s,t)\)-connectivity [Rei08]. This problem was actually easy for constant degree expanders, but of course the input graph is generally not a constant degree expander. The goal becomes to implicitly convert the graph into a constant degree expander. There we iterated between powering the graph — amplifying the spectral gap — and taking a zig-zag product with a constant degree expander — sparsifying the graph. Together these operations formed a net gain in the spectral gap, while keeping the degree constant. A logarithmic number of iterations transformed the input graph (implicitly) into a constant degree expander.

The proof of Dinur [Din07] is similar in spirit, trying to amplify a graph CSP by a series of transformations. In fact Dinur mentions deterministic log-space connectivity as an inspiration. Here we have three basic transformations:

1. **Expander-ification**, where we make the underlying graph a constant degree expander.

2. **Error amplification by powering**, where we amplify the unsat of the graph by taking a power of the graph and creating new constraints appropriately.

3. **Alphabet reduction.** Where we reduce the alphabet size of the graph CSP (which increases in the powering step).

Each of these three steps take as input one graph CSP \(G\) and outputs another, \(G'\). As we analyze these steps, we will be careful to ensure the following critical properties.

1. **Completeness:** If \(\text{unsat}(G) = 0\), then \(\text{unsat}(G') = 0\).

2. **Soundness:** \(\text{unsat}(G') \geq \beta \text{unsat}(G)\) for some value \(\beta > 0\).

For soundness, we want \(\beta > 1\), but an individual operation may have \(\beta < 1\) if it is principally concerned with managing other parameters (such as the alphabet size, or the degree). We will show that the three steps together gives \(\beta > 1\), for a fixed constant \(\beta\), while keeping the other parameters under control (and more precisely, universal constants).

**Expander-ification.** Let us break down these steps in a little more detail. Below The first step, “expander-ification”, preprocesses \(G\) so that it is a constant degree expander. The constant degree is important because the subsequent power step has to blow up the size of the alphabet exponentially in the degree. The spectral gap is important for the alphabet reduction step after that.
Lemma 23.5. There exists universal constants $d \in \mathbb{N}$, $\gamma > 0$, and $\beta_1 > 0$ for which the following holds. Given an instance of graph CSP $G$, one can compute an instance of graph CSP $G'$ with the following properties.

1. The graph supporting $G'$ is a $d$-regular undirected graph.
2. $G'$ has the same alphabet size as $G$.
3. $\beta_1 \text{unsat}(G) \leq \text{unsat}(G') \leq \text{unsat}(G)$.
4. The spectral gap on $G'$ is $\geq \gamma$.
5. The size of $G'$ is at most a constant factor greater than the size of $G$.

We prove lemma 23.5 in section 23.4.

Error Amplification. The next step, powering, is where we amplift UNSAT. Recall that in normal graphs, taking the $k$th power means we generate an edge for every $k$-edge walk in the input graph. The resulting graph is denser and the spectral gap is larger.

Powering a graph CSP is similar in spirit but more complicated since we must address the CSP aspects as well. Here we give a high level sketch. The underlying graph will be the $k$th graph power. Each edge $\{u, v\}$ in the powered graph corresponds to a $k$-edge walk in the original graph, which corresponds to $k$ binary constraints. Loosely speaking, these $k$ constraints are combined into one large constraint that in some sense requires all $k$ constraints to be satisfied simultaneously. Intuitively, this increases the unsatisfiability since we now have to simultaneously satisfy $k$ clauses in the input graph CSP to satisfy just 1 clause in the powered graph CSP.

For this idea to make syntactic sense, we have to increase the alphabet so that for each “power constraint” of $k$ input clauses, each vertex has “$k$ letters” to be supplied to each of the clauses. Thus the alphabet expands from $A$ to $A^{d^k}$; i.e., $d^k$ letters per vertex. These $d^k$ labels for a vertex $v$ are interpreted as labeling the entire $k$-step neighborhood of $v$.

Again, this is only a high-level description, and we explain the construction much more carefully in section 23.5. All put together, the bundled constraints ramp up UNSAT, but we pay the price of a much larger alphabet size.

Lemma 23.6. Let $d$, $\gamma$, $|A|$ be fixed. Then there exists $\beta_2 > 0$ for which the following holds for all $t = 2s + 1$ where $s \in \mathbb{N}$. Let $G$ be a regular graph CSP over an alphabet $A$ with degree $d$ and spectral gap (at least) $\gamma$. Then one can compute a graph-CSP $G^t$ with the following properties.
1. $G^t$ is regular with degree $d^t$, alphabet $A^{d^t}$, 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 $\gamma$ in the parameter $\beta_2$.

**Alphabet reduction.** The third step addresses the blow up in alphabet size from powering the graph CSP.

**Lemma 23.7.** 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

$$\beta_3 \text{unsat}(G) \leq \text{unsat}(G') \leq \text{unsat}(G).$$

**Putting it all together.** All put together, these three operations take as input one graph CSP $G$ over a constant-sized alphabet $A_0$ and outputs another graph CSP $G'$ over the same alphabet. For a parameter $t \in \mathbb{N}$, if $\text{unsat}(G)$ is a universal constant factor smaller than $t$, then $\text{unsat}(G')$ is at most a universal constant factor
smaller than $\sqrt{t} \text{unsat}(G)$. (See fig. 23.1.) By making $t$ a sufficiently small constant, this means that if $\text{unsat}(G)$ is smaller than a universal constant, than $\text{unsat}(G')$ is greater than $\text{unsat}(G)$ by a universal constant $> 1$. That is, we’ve successfully amplified $\text{unsat}$ without increasing the input size.

### 23.4 Expander-ification

The first transformation preprocesses the graph-CSP to be a constant degree expander. We proceed in two stages where the first stage addresses the degree and the second stage addresses the spectral gap.

To make the degree small and uniform, we replace each vertex $v \in V$ with a low degree expander as follows.

1. For each vertex $v$ and each edge $e$ incident to $v$, we create a new vertex $(v, e)$.
2. For each edge $e = (u, v)$, we have an edge from $(u, e)$ to $(v, e)$ with the same constraint as $e$.
3. Fix $v$. We currently have an auxiliary vertex $(v, e)$ for every edge $e$ incident to $v$. Let $G_0$ be a $d_0$-regular expander with vertex set $\{(v, e)\}$. For each edge $e$ in the expander, we assign the “equality constraint” $C_e = \{(a, a) : a \in A\} \subset A^2$.

**Exercise 23.3.** Let $G'$ be the graph CSP obtained from $G$ via steps 1–3 above. Prove that $G'$ has the following properties:

(a) The total number of edges of $G'$ is within a constant factor of the number of edges of $G$.
(b) $G'$ has the same alphabet as $G$.
(c) $G'$ is a $d$-regular graph for a universal constant $d$.
(d) $c \text{unsat}(G) \leq \text{unsat}(G') \leq \text{unsat}(G)$ for some universal constant $c$.

We now assume that $G = (V, E)$ is regular with constant degree $d = O(1)$. We now make $G$ an expander while retaining the degree. This is done by simply overlaying

Let $H = (V, E_H)$ be a constant degree expander on the same vertex set $V$. For all $e \in E_H$, let $C_e$ be the trivial constraint that allows for all pairs of labels. Let $G' = (V, E + E_H)$ be the graph-CSP combining the clauses from $G$ and $H$. The $G'$ is an expander while retaining the salient properties of $G$, which is left to the reader to show.
Exercise 23.4. Prove the following properties about the graph-CSP $G' = (V, E + E_H)$ described on the previous page.

(a) $G'$ has constant degree.
(b) $G'$ has constant spectral gap $\gamma$.
(c) The total size of $G'$ is at most a constant factor greater than $G'$.
(d) $\Omega(\text{unsat}(G)) \leq \text{unsat}(G') \leq \text{unsat}(G)$.

Combining our two preprocessing steps gives lemma 23.5.

23.5 Error amplification

We now turn to the error amplification stage. The input is a graph-CSP $G$ where the underlying graph is a regular expander with constant degree $d$. The goal is to produce a new 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 [Din07] offers the following helpful intuition. Fix an assignment $\pi : V \to A$. Suppose we sample $t$ edges in $G$ uniformly at random and test if any of the corresponding constraints are not satisfied. The probability that at least one of them is unsatisfied is $1 - (1 - \text{unsat}(G))^t \approx t \text{unsat}(G)$ (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 $1 - (1 - \text{unsat}(G))^t \approx t \text{unsat}(G)$, which is good. However, (a) the resulting CSP is no longer binary, and (b) the number of constraints in the CSP increases substantially from $m$ to roughly $mt$.

In short, 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 behaves similarly to uniform sampling. Here (the underlying graph of) $G$ is a constant degree expander. Since $G$ is an expander, then the $t$-step walks should behave like independent samples of $t$ edges. Thus we create a graph-CSP instance on top of the $t$-th graph power of $G$.

Powering a graph-CSP. We sketched the construction in section 23.3 and now we describe it in full detail. Consider the graph $G^t$ that has edges corresponding to lazy $t$-step random walks. More precisely, let us generate for each vertex $v$, $d$ self-loops at $G$. Call this graph $G_L$: a random $t$-step walk in $G_L$ corresponds to a lazy $t$-step walk in $G$. We create an edge $(v_0, v_t)$ in $G^t$ for every $t$-step walk $(v_0, \ldots, v_t)$ in $G_L$. 

319
For a vertex \( v \), let \( N^t(v) \) denote the set of vertices within \( t \) steps of \( v \) in \( G \) (including \( v \)). We have
\[
|N^t(v)| \leq D \text{ where } D = \sum_{i=0}^{t} d^i = (1 + d)^t.
\]

We use the alphabet \( A^D \). We identify an \( A^D \)-label \( \tilde{\pi}(v) \in A^D \) as an \( A \)-labeling of all \( N^t(b) \). We let \( \tilde{\pi}(v, w) \in A \) denote the label assigned by \( \tilde{\pi}(v) \) to \( w \in N^t(v) \).

For each \( t \)-step walk \( w = (v_0, \ldots, v_t) \) in \( G_L \), which corresponds to a distinct edge \( e_w \) between \( v_0 \) and \( v_t \) in \( G' \), we create a constraint \( C_w \) that is satisfied by \( \tilde{\pi}(v_0), \tilde{\pi}(v_t) \in A^D \) iff the following holds.

(a) For each edge \( e = (v_i, v_{i+1}) \in w \), the labels \( (\tilde{\pi}(v_0, v_i), \tilde{\pi}(v_t, v_{i+1})) \in A^2 \) satisfies the constraint \( C_e \) in \( G \).

(b) For all \( i \), \( \tilde{\pi}(v_0, v_i) = \tilde{\pi}(v_t, v_i) \).

Property (a) is where we must satisfy all \( t \) underlying constraints. Property (b) forces \( v_0 \) and \( v_t \) to agree on all the vertices along the walk.

**Analysis.** It is easy to see that a satisfying assignment for \( G \) gives a satisfying assignment for \( G' \). 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 \( \tilde{\pi} : V \to A^D \), we derive a labeling \( \pi : V \to A \) and show that \( \text{UNSAT}(\pi | G) \leq \text{poly}(1/t) \text{UNSAT}(\tilde{\pi} | G') \). The labeling \( \pi \) is derived from \( \tilde{\pi} \) as follows.

Fix \( v \in V \). Let \( v_0 = v, v_1, \ldots \) denote a lazy random walk starting from \( v \). Consider the random label \( \tilde{\pi}(v_{t/2}, v) \). We define \( \pi(v) \) to be the most likely label to arise as \( \tilde{\pi}(v_{t/2}, v) \). We have
\[
P[\tilde{\pi}(v_{t/2}, v) = \pi(v)] \geq 1/|A|, \quad (23.1)
\]
which as far as we’re concerned, is at least a constant. Up to constants, (23.1) extends to indices close to \( t/2 \). Let \( I = \{i : |i - t/2| \leq \sqrt{t/2} \} \).

**Lemma 23.8.** For sufficiently large \( t \), there exists a universal constant \( c > 0 \) such that for all \( i \in I \),
\[
P[\tilde{\pi}(v_i, v) = \pi(v)] \geq c/|A|.
\]
We briefly sketch the intuition and postpone the formal proof to the end of the section. Because all $i \in I$ are very close to $t/2$, the number of “non-lazy” steps $j$ in an $i$-step random walk is distributed almost the same for all $i \in I$. Conditional on $j$, $\pi(v_i, v)$ is independent of $i$. Putting these two facts together means that $\pi(v_i, v)$ is distributed very similarly for all $i$, and in particular, similar to $\pi(v_{t/2}, v)$.

Before proceeding to the main part of the proof, we require one more technical lemma that brings into play the assumption that $G$ is an expander. Recall that the edges along a random walk in an expander behave almost independently. This is reflected in the following lemma.

**Lemma 23.9.** Let $F \subset E$ be any subset of edges, and let $\mu = |F|/|E|$. Consider a random walk $v_0, v_1, v_2, \ldots$ in $G$, where initially $v_0$ is a uniformly random endpoint of a uniformly random random edge from $F$. Then

$$P[(v_i, v_{i+1}) \in F] \leq \mu + (1 - \gamma)^i,$$

for all $i$, where $\gamma$ is the spectral gap of $G$.

We postpone the proof of this lemma as well until the end of the section, and instead turn to the main claim. In the following, note that we will take $t$ to be a constant, so if UNSAT($\pi | G$) $\geq 1/t$ then we have already accomplished our main goal.

**Lemma 23.10.** $\text{UNSAT}(\bar{\pi} | G^t) \geq \Omega\left(\sqrt{t}\right) \min\{\text{UNSAT}(\pi | G), \frac{1}{t}\}$.

**Proof.** Let $F \subset E$ be the set of edges failed by $\pi$, and let $\mu = |F|/|E| = \text{UNSAT}(\pi | G)$. If $\mu \geq 1/t$, then drop edges from $F$ until $\mu \leq 1/t$. We have $1/t \geq \mu \geq \Omega(1) \min\{\text{UNSAT}(\pi | G), \frac{1}{t}\}$.

Let $w = (v_0, \ldots, v_t)$ be a lazy $t$-step walk in $G$ sampled uniformly at random. For each $i \in I$, we define a random indicator variable $X_i \in \{0, 1\}$ where $X_i = 1$ if, letting $e_i = (v_{i-1}, v_i)$ be the $i$th edge in $w$,

(a) The $i$th edge in $w$, $e_i = (v_{i-1}, v_i)$, has labels $\bar{\pi}(v_{i-1}, v_0) = \pi(v_{i-1})$ and $\bar{\pi}(v_t, v_i) = \pi(v_i)$.

(b) Either $e_i$ is a loop or $\pi$ fails the constraint $C_{e_i}$.

Let $X = \sum_{i \in I} X_i$. Observe that $\text{UNSAT}(G^t) \geq P[X > 0]$. We make two claims about $X$.

Claim 1. For all $i \in I$, $\mathbb{E}[X_i] \geq \Omega(1)\mu$.

Claim 2. $\mathbb{E}[X^2] \leq O\left(\sqrt{t}\right)\mu$. 
We will prove the claims later. Let us first assume they hold and complete the proof. Linearity of expectation and claim 1 gives $\mathbb{E}[X] \geq \Omega(\sqrt{t}) \mu$. We also have

$$\mathbb{E}[X^2] = \mathbb{E}[X \mid X > 0]^2 \mathbb{P}[X > 0] \leq \mathbb{E}[X^2 \mid X > 0] \mathbb{P}[X > 0] = \mathbb{E}[X^2] \mathbb{P}[X > 0]$$

by (a) Jensen’s inequality (for the convex function $f(x) = x^2$). Rearranging and applying the claims, we have

$$\mathbb{P}[X > 0] \geq \frac{\mathbb{E}[X^2]}{\mathbb{E}[X] \mathbb{P}[X > 0]} \geq \Omega(\sqrt{t}) \mu,$$

as desired.

It remains to prove the claims, starting with claim 1. Suppose we sample $X_i$ alternatively as follows. Sample an edge $(v_{i-1}, v_i)$ uniformly at random. Take lazy random walks $v_{i-1}, \ldots, v_0$ and $v_i, \ldots, v_t$, and output the walk $w = (v_0, \ldots, v_t)$. This produces a uniformly random lazy walk because $G$ is regular. We have

$$\mathbb{P}[X_i = 1] = \mu \cdot \mathbb{P}[^\pi(v_0, v_{i-1}) = \pi(v_{i-1})] \mathbb{P}[^\pi(v_i) = \pi(v_i)].$$

Recall from lemma 23.8 that, for $i \in I$, the marginal probabilities of $^\pi(v_i)$ are within a constant factor of the marginal probabilities of $^\pi(v_i, v_{i/2})$, which in turn is at least $1/|A|$. Similarly for $^\pi(v_0, v_i)$. Thus

$$\mathbb{P}[X_i = 1] \geq \Omega(1) \mu / |A|^2 = \Omega(1) \mu,$$

as desired.

The remaining claim, claim 2, is about the variance of $X = \sum_{i \in I} X_i$. To this end we have an intermediate claim analyzing the cross-terms $X_i X_j$.

**Claim 3.** Let $i, j \in I$ with $i \neq j$. Then $\mathbb{E}[X_i X_j] \leq \mu \left( \mu + (1 - \gamma)^{j-i-1} \right)$.

Let us define random indicators variables $Y_i, Y_j \in \{0, 1\}$ that indicate whether the $i$th edge is in $F$. Then $0 \leq X_i \leq Y_i$ hence $\mathbb{E}[X_i X_j] \leq \mathbb{E}[Y_i Y_j]$. Write

$$\mathbb{E}[Y_i Y_j] = \mathbb{E}[Y_i] \mathbb{E}[Y_j \mid Y_i = 1] = \mu \mathbb{E}[Y_j \mid Y_i = 1].$$

The remaining term, $\mathbb{E}[Y_j \mid Y_i = 1]$, is equivalent to the probability that a random walk starting at a random endpoint of a uniformly random edge in $F$ takes its $(j-i)$th step in $F$. By lemma 23.9, this probability is at most $\mu + (1 - \gamma)^{j-i-1}$.
Now we prove claim 2. We have
\[
E[X^2] = \sum_{i \in I} E[X_i] + 2 \sum_{i,j \in I, i < j} E[X_i X_j]
\leq \mu |I| + 2 \mu^2 |I|^2 + 2 \mu \sum_{i,j \in I, i < j} (1 - \gamma)^{j-i-1}
\overset{(b)}{=} \Omega(1) \mu |I| + 2 \mu^2 |I|^2 \overset{(c)}{\leq} \Omega(\sqrt{t}) \mu.
\]

Here (b) is by claim 3. (c) is because \( \mu \geq 1/t \) and \( |I| = O(\sqrt{t}) \).

This completes the proof of lemma 23.10. \( \square \)

This establishes, modulo lemma 23.8 and lemma 23.9 which were introduced earlier in the section. The remainder of this section is devoted to proving these lemmas.

**An expander mixing lemma for edges.** Let us first prove lemma 23.9 since it is arguably more interesting. In particular, it reveals why it is important that \( G \) is an expander graph. We briefly recall the motivation. We want to argue that the failed edges \( F \) are not too correlated along random walks. This is because if they are correlated, then the number of bad edges per walk, \( X \) in our high analysis, will have high variance (and in particular, claim 2 will fail).

**Lemma 23.9.** Let \( F \subset E \) be any subset of edges, and let \( \mu = |F|/|E| \). Consider a random walk \( v_0, v_1, v_2, \ldots \) in \( G \), where initially \( v_0 \) is a uniformly random endpoint of a uniformly random edge from \( F \). Then
\[
P[(v_i, v_{i+1}) \in F] \leq \mu + (1 - \gamma)^i.
\]
for all \( i \), where \( \gamma \) is the spectral gap of \( G \).

**Proof.** Let \( G \) be a regular undirected graph and let \( R \) be the random walk map. Let \( x \in \Delta V \) be the initial distribution for \( v_0 \). For each vertex \( v \), we have
\[
x(v) = \frac{\text{(# edges in } F \text{ incident to } v)}{2|F|}.
\]
Note that \( x(v) \leq (d/2|F|) \) for all \( v \). Define \( y : V \to [0,1] \) be setting, for each \( v \in V \),
\[
y(v) = \frac{\text{(# edges in } F \text{ incident to } v)}{d} = \left( \frac{2|F|}{d} \right) x(v).
\]

This is equal to the probability that a random step from \(v\) is in \(F\). The probability that the \(t\)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 \(G\) is a regular undirected graph with spectral gap \(\gamma\), we can write

\[
R^{t-1} = \frac{1}{n}(\mathbb{1} \otimes \mathbb{1}) + R'
\]

where \(\|R'\| \leq \gamma^{t-1}\). The remainder of the proof is left to the reader in the following exercise.

\[\square\]

**Exercise 23.5.** Complete the proof above via the following steps.

1. Prove that \(\langle x, R^{t-1} x \rangle \leq \frac{1}{n} + \frac{\gamma^{t-1} d}{2|F|}\).
2. Prove that \(P[(v_{t-1}, v_t) \in F] \leq \frac{|F|}{|E|} + \gamma^{t-1}\).

**Similarity of lazy random walks.** The final lemma to prove is about how lazy \(i\)-step walks act similarly for different \(i \in I\). We start by stating the following lemma about binomial distributions.

**Lemma 23.11.** For every \(c > 0\), there exists some constant \(z \in (0, 1)\) and \(n_0\) such that, if \(n_0 < n - \sqrt{n} \leq m < n + \sqrt{n}\), then for all \(k\) such that \(|k - n/2| \leq c\sqrt{m}\), we have:

\[
z \leq \frac{P[B_n = k]}{P[B_m = k]} \leq \frac{1}{z}
\]

The proof of lemma 23.11 is given as exercise 23.7. Let us continue to prove lemma 23.8.

**Lemma 23.8.** For sufficiently large \(t\), there exists a universal constant \(c > 0\) such that for all \(i \in I\),

\[
P[\pi(v_i) = \pi(v)] \geq c/|A|.
\]

**Proof.** For \(i \in \mathbb{N}\), let \(B_i\) be the number of non-lazy steps out of the first \(k\) steps in the lazy random walk. Choose \(c > 0\) such that the probability that \(|B_{i/2} - t/4| \geq c\sqrt{t} \leq 1/2|A|\). For ease of notation, let \(E_i\) be the event that \(\pi(v_i | v_0) = \pi(v_0)\). We want to
23.6 Alphabet reduction

Figure 23.2: A high-level schematic for the alphabet reducing transformation in section 23.6, on a triangle graph.

show that for $i \in I$, we have $\mathbf{P}[E_i] \geq \Omega(1) \mathbf{P}[E_{t/2}]$. Let $J = \left\{ j : |j - t/4| \leq c \sqrt{t/2} \right\}$.

We have

\[
\mathbf{P}[E_i] \geq \sum_{j \in J} \mathbf{P}[E_i \mid B_i = j] \mathbf{P}[B_i = j]
\]

\[
\geq z \sum_{j \in J} \mathbf{P}[E_i \mid B_i = j] \mathbf{P}[B_{t/2} = j]
\]

\[
\overset{(a)}{=} z \sum_{j \in J} \mathbf{P}[E_{t/2} \mid B_{t/2} = j] \mathbf{P}[B_{t/2} = j]
\]

\[
\overset{(b)}{=} z \left( \mathbf{P}[E_{t/2}] - \frac{1}{2|A|} \right) \overset{(c)}{=} \frac{z}{2} \mathbf{P}[E_{t/2}],
\]

as desired. Here (a) is by lemma 23.11, for $t$ sufficiently large and $z$ the constant asserted in lemma 23.11. The reason for (b) is left as an exercise below. (c) is by choice of $c$. (d) is because $\mathbf{P}[E_{t/2}] \geq 1/|A|$.

Exercise 23.6. Justify equality (b) in the proof of lemma 23.8.

23.6 Alphabet reduction

This section is about the third graph-CSP transformation reduction, where the goal is to reduce the size of the alphabet. We take as input a graph CSP $G$ with alphabet $A$. Our goal is to reduce the alphabet to an alphabet $A_0$ where $|A_0|$ is a universal constant. (In fact, the full details reveal that $|A_0| = 8$.)

Step 1: Reduce the alphabet size to 2 with an error correcting code. Let $\mathcal{C} : A \to \{0, 1\}^\ell$ be an error correcting code with $\ell \leq O(\log(|A|))$ and relative distance $\rho \in [0, 1]$; that is, for any distinct $a_1, a_2 \in A$, the encodings $\mathcal{C}(a_1), \mathcal{C}(a_2) \in \{0, 1\}^\ell$
Randomized Proofs and Verification by Random Walks

23.6. Alphabet reduction

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 \(f_e\).

**Step 2: Replace each hyperedge constraint with a graph CSP.** The next step converts the hypergraph generated above into a graph. We will apply a certain operation hyperedge-wise that reduces each hyperedge independently to a graph CSP. Analyzing this operation requires new (Fourier-analytic) techniques that are better to discuss separately (chapter 24). For now we will just use it as a black box.

Let \(\epsilon_0 > 0\) be a parameter \(A_0\) be a finite alphabet of cardinality to be determined. For each formula \(f_e\), we construct a graph-CSP \(G_e = (V_e, E_e, A_0, \{C_{e,f} : f \in E_e\})\), where \(V_u, V_v \subseteq V_e\), and which has the following properties.

1. **Completeness:** If \(f_e(x_u, x_v) = 1\), then we can extend \((x_u, x_v) : V_u \cup V_v \to \{0, 1\}\) to an assignment \(\sigma : V_e \to A_0\) that satisfies all of \(G_e\).

2. **Soundness:** If \(f_e(x_u, x_v) = 0\), then any extension \(\sigma : V_e \to A_0\) of \((x_u, x_v)\) has error proportional to the relative Hamming distance between \((x_u, x_v)\) and \(F_e\).

\[
\text{UNSAT}(\sigma \mid G_e) \geq \epsilon_0 \text{ dist}((x_u, x_v), F_e)
\]

for all extensions \(\sigma\) of \((x_u, x_v)\), where \(\text{dist}(a, b)\) denotes the relative Hamming distance between \(a\) and \(b\).

Moreover, the parameter \(\epsilon_0\) and the size of \(A_0\) depending only on \(|\ell|\). The size of \(|E_e|\) is the same for all \(e\).

In lieu of a proper analysis, we provide some high-level comments. For each edge \(e\), we transform a boolean function on \(2\ell\) 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 locally, we are still blowing up a constant to a constant.

The above construction, that is applied to each boolean function, can be understood as an inefficient and distance-preserving PCP.

**Analysis.** Thus we have a two-step process that takes one graphical CSP and produces another with a constant alphabet size. Between the two steps we have a hypergraph CSP over the alphabet $\{0, 1\}$. We assume that step 2 is implementable, and defer those details to chapter 24.

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 23.12.** $\operatorname{unsat}(\bar{G}) \leq \operatorname{unsat}(G)$.

**Proof.** Let $\pi : V \rightarrow A$ be an assignment that attains $\operatorname{unsat}(G)$. 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 $\operatorname{unsat}(\bar{G}) \leq \operatorname{unsat}(G)$. \qed

**Lemma 23.13.** For $\beta_3 = \epsilon_0 \rho/4$, we have

$$\beta_3 \operatorname{unsat}(G) \leq \operatorname{unsat}(\bar{G}).$$

**Proof.** Let $\bar{\pi} : \bar{V} \rightarrow A_0$ be an assignment for $\bar{\pi}$ that attains $\operatorname{unsat}(\bar{G})$. Let $\pi : V \rightarrow A$ be the assignment defined by decoding; for each vertex $v$, we have

$$\pi(v) = D(\bar{\pi}(v_1), \ldots, \bar{\pi}(v_{\ell})).$$

We claim that for each edge $e$ that is not satisfied by $\pi$, a $\beta_3$-fraction of $G_e$ is unsatisfied for some value $\beta$ that depends on $|A_0|$ and $\rho$. It then follows that $\operatorname{unsat}(\bar{G}) \geq \beta_3 \operatorname{unsat}(G)$. 327
Let \( e = (u, v) \) be an edge that is not satisfied by \( \pi \). For ease of notation, let us denote
\[
\bar{u} \overset{\text{def}}{=} (\bar{u}_1, \ldots, \bar{u}_\ell) \quad \text{and} \quad \bar{\pi}(\bar{u}) \overset{\text{def}}{=} (\bar{\pi}(u_1), \ldots, \bar{\pi}(u_\ell)).
\]
Since \((\pi(u) = D(\bar{u}), \pi(v) = D(\bar{v}))\) 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 \( G_e \).

### 23.7 Additional notes and materials

See [AB09, Chapter 11] for further background on the PCP theorem.

**Lecture materials, part 1.** Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

**Lecture materials, part 2.** Click on the links below for the following files:
- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Recorded video lecture.

### 23.8 Exercises

**Exercise 23.1.** Prove that \( \text{PCP}(O(\log n), O(1)) \subseteq \text{NP} \).

**Exercise 23.2.** Prove theorem 23.1 and theorem 23.3 are equivalent. Below we give part of the proofs, in both directions, to get you started.

1. **Theorem 23.1 \implies theorem 23.3.** Suppose the PCP theorem, theorem 23.1, 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...

2. Theorem 23.3 \( \implies \) theorem 23.1. 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...

Exercise 23.3. Let \( G' \) be the graph CSP obtained from \( G \) via steps 1–3 on page 318. Prove that \( G' \) has the following properties:
(a) The total number of edges of \( G' \) is within a constant factor of the number of edges of \( G \).
(b) \( G' \) has the same alphabet as \( G \).
(c) \( G' \) is a \( d \)-regular graph for a universal constant \( d \).
(d) \( c \text{unsat}(G) \leq \text{unsat}(G') \leq \text{unsat}(G) \) for some universal constant \( c \).

Exercise 23.4. Prove the following properties about the graph-CSP \( G' = (V,E+E_H) \) described on page 318.
(a) \( G' \) has constant degree.
(b) \( G' \) has constant spectral gap \( \gamma \).
(c) The total size of \( G' \) is at most a constant factor greater than \( G' \).
(d) \( \Omega(\text{unsat}(G)) \leq \text{unsat}(G') \leq \text{unsat}(G) \).

Exercise 23.5. Complete the proof above via the following steps.
1. Prove that \( \langle x, R^t-1x \rangle \leq \frac{1}{n} + \frac{\gamma^{t-1}d}{2|F|} \).
2. Prove that \( P[(v_{t-1}, v_t) \in F] \leq \frac{|F|}{|E|} + \gamma^{t-1} \).

Exercise 23.6. Justify equality (b) in the proof of lemma 23.8.

Exercise 23.7. For \( n \in \mathbb{N} \), let \( B_n \) be a binomially distributed random variable with probability \( p = 1/2 \). Prove the following.
For every $c > 0$, there exists some constant $z \in (0, 1)$ and $n_0$ such that, if $n_0 < n - \sqrt{n} \leq m < n + \sqrt{n}$, then for all $k$ such that $|k - n/2| \leq c\sqrt{m}$, we have

$$z \leq \frac{\Pr\{B_n = k\}}{\Pr\{B_m = k\}} \leq \frac{1}{z}$$
A boolean function is a function $f : \{0, 1\}^n \to \mathbb{R}$ that takes as input a sequence of bits and outputs a single value - often another bit.\footnote{Of course one can consider functions that output more than one value - but real-valued boolean functions suffice for the current discussion. In fact this note only really requires boolean functions of the form $f : \{0, 1\}^n \to \{-1, 1\}$.} Clearly any (deterministic) program making a binary decision is a boolean function, which makes boolean functions a natural object of study. Here we explore a testing approach that takes a boolean function $f$ as a black box, queries $f$ at a limited number of inputs, and analyzes the outputs to make useful statements about $f$.

The universality of boolean functions makes them attractive to study. But the same universality makes it seem rather daunting to be able to obtain concrete and useful observations about them. Nonetheless today we will see a few interesting things that one can do, at least approximately, by combination of randomization and an appropriate change of basis.

Today we will discuss a few introductory topics in property testing, which takes as input $f$ and tries to decide if $f$ has a certain property. We would only be able to do so approximately, and differentiate functions that have the property (exactly) from functions that fail to have the property for a constant fraction of the inputs. For example, we will show how to approximately test boolean functions for the following properties.

- **Linearity:** whether $f : \{0, 1\}^n \to \{0, 1\}$ satisfies $f(x + y) = f(x) + f(y)$ for all $x$ and $y$.
- **Dictatorship:** whether $f : \{0, 1\}^n \to \{-1, 1\}$ is of the form $f(x) = (-1)^{x_i}$ for some $i \in [n]$. 

\footnote{Of course one can consider functions that output more than one value - but real-valued boolean functions suffice for the current discussion. In fact this note only really requires boolean functions of the form $f : \{0, 1\}^n \to \{-1, 1\}$.}
The main goal of today’s discussion is to describe the following universal tester for proof systems.

**Theorem 24.1.** Let \( L \subseteq \{0,1\}^n \). Let \( p = 2^{2^n} \). Then there is a randomized algorithm \( A_L : \{0,1\}^n \times \{0,1\}^p \rightarrow \{0,1\} \) that, given oracle access to \( x \in \{0,1\}^n \) and \( y \in \{0,1\}^p \), has the following properties.

(a) \( A_L \) makes 3 queries to bits of \( x \) or \( y \).

(b) If \( x \in L \), then there exists \( y \in \{0,1\}^p \) such that \( A_L(x,y) = 1 \) always.

(c) If \( x \notin L \), then for all \( y \in \{0,1\}^p \), we have

\[
P[A_L(x,y) = 0] \geq 0.001 \min_{y \in L} \frac{\|x - y\|_0}{n}.
\]

The connection between theorem 24.1 to boolean functions is not self-evident. Let us briefly describe the algorithm underlying theorem 24.1 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

\[
\chi_i(x) = \begin{cases} 
1 & \text{if } x_i = 0 \\
-1 & \text{if } x_i = 1 
\end{cases}
\]

\( \chi_i \) is called a dictator function and is the topic of section 24.4. 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 as \( (2^N = 2^{2^n}) \)-dimensional \( \{-1,1\} \)-vectors. We will create a test that, given \( x \in \{0,1\}^n \) and \( y \in \{0,1\}^p \), simultaneously tests

(a) If \( y \in D_L \).

(b) Conditional on \( y \in D_L \), if \( y = \chi_x \) (where we identify \( x \) with an index in \( N \)).

Thus the theorem reduces to understanding two tests about Boolean functions. We will build these tools over the course of this note and will finish with the proof of theorem 24.1.

To motivate theorem 24.1, recall that an important ingredient of the PCP theorem (chapter 23) was a subroutine that took as input a constant-size boolean function and output a graph CSP that modeled them in an error-preserving fashion. Let us now show how to obtain this result using the universal tester above.
Theorem 24.2. Given a language $L \subset \{0,1\}^n$, one can construct a graph-CSP with graph $G_L = (V_L, E_L)$, alphabet $A$, and constraints $\{C_e \subset A^2 : e \in E_L\}$ with the following properties.

1. $A = \{0,1,\ldots,7\}$.

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 $\text{unsat}(\sigma|_{G_L}) = 0$.

   • 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

     $$\text{unsat}(\sigma|_{G_L}) \geq .001 \min_{y \in L} \|x-y\|_0 \frac{n}{n}.$$ 

Proof. Fix a universal tester $T$ for the language $L$ that takes as input $x \in \{0,1\}^n$ and $y \in \{0,1\}^p$, for $p = 2^{2^n}$. We create a vertex for each of the following.

1. For each $i \in [n]$, a vertex $X_i$ (modeling the input bit $x_i$).

2. For each $i \in [p]$, a vertex $Y_i$ (modeling the proof bit $y_i$).

3. The universal tester $T$ flips a finite number of coins. For every possible outcome of coin tosses $\omega$, 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 $Y_{k_i}$.
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 $L$.

**Exercise 24.1.** Complete the proof of theorem 24.2 by verifying that the graph-CSP described above satisfies the claimed properties.

24.2 Fourier analysis of boolean functions

A *boolean function* is a real-valued function defined over bit-strings of a fixed length; i.e.,

$$f : \{0, 1\}^n \to \mathbb{R}.$$ 

Note that sums of boolean functions and rescaled boolean functions are again boolean functions. In particular, we can identify the set of all boolean functions with the $2^n$-dimensional Euclidean vector space $\mathbb{R}^{\{0,1\}^n}$. Here the $i$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 $\langle \cdot, \cdot \rangle_b$:

$$\langle f, g \rangle_b \overset{\text{def}}{=} \frac{1}{2^n} \langle f, g \rangle = \mathbf{E}_{x \sim \{0,1\}^n} [f(x)g(x)],$$

where in the RHS $x$ is sampled uniformly from $\{0,1\}^n$. Let

$$\|f\|_b = \sqrt{\langle f, f \rangle_b} = \sqrt{\mathbf{E}_x [f^2(x)]}$$

denote the corresponding norm. This norm rescales the standard Euclidean norm by $2^{-n/2}$. Here are a couple helpful identities to get us started.

**Lemma 24.3.** For two boolean functions $f, g : \{0,1\}^n \to \{0,1\}$, we have

$$\|f - g\|_b^2 = \mathbf{P}_x [f(x) \neq g(x)].$$

**Lemma 24.4.** For two boolean functions $f, g : \{0,1\}^n \to \{-1, 1\}$, we have

$$\mathbf{P}[f(x) \neq g(x)] = \frac{1}{4} \|f - g\|_b^2.$$
Lemma 24.5. For any boolean function \( f : \{0, 1\}^n \rightarrow \{-1, 1\} \), we have \( \|f\|_b^2 = 1 \).

We leave the proofs of the above as exercises.

So far we have expressed boolean functions in terms of their “truth tables” as vectors in \( \mathbb{R}^{\{0,1\}^n} \), but of course there are many possible bases over \( \mathbb{R}^{\{0,1\}^n} \) that one could work with. Fourier analysis is based on the following choice of basis. For each set \( S \subseteq [n] \), define a boolean function \( \chi_S : \{0, 1\}^n \rightarrow \{-1, 1\} \) by
\[
\chi_S(x) = (-1)^{\sum_{i \in S} x_i} = \begin{cases} 1 & \text{if } \sum_{i \in S} x_i \text{ is even}, \\ -1 & \text{if } \sum_{i \in S} x_i \text{ is odd}. \end{cases}
\]

We call \( \chi_S \) the \( S \)th Fourier basis function; sometimes \( \chi_S \) is called the parity function over \( S \). The Fourier basis functions have many convenient properties of which we list a few. For \( S = \emptyset \), we have \( \chi_\emptyset = 1 \), the all-one’s vector. For all \( S, T \subseteq [n] \), we have
\[
\chi_S \chi_T = \chi_{S \Delta T},
\]
where \( S \Delta T = (S \cup T) \setminus (S \cap T) \) denotes the symmetric difference. We also have, for all nonempty sets \( S \neq \emptyset \),
\[
\mathbb{E}_{x \in \{0, 1\}^n} [\chi_S(x)] = 0.
\]

The above is easy to see for singleton sets \( S = \{i\} \). For general sets \( S \) and \( S' = S - i \), we have
\[
\mathbb{E}[\chi_S] \equiv \mathbb{E}[\chi_{S'}(x) \chi_i(x)] \equiv \mathbb{E}[\chi_{S'}(x)] \mathbb{E}[\chi_i(x)] \equiv 0.
\]
Here (a) is by (24.1). (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 \( S, T \subseteq [n] \).

Exercise 24.2. Verify eqs. (24.1) and (24.2) above.

Equation (24.2) means that the functions \( \{\chi_S : S \subseteq [n]\} \) form an orthonormal set. There are also \( 2^n \) many of them, and we are working in a \( 2^n \)-dimensional space, so in fact they form an orthonormal basis (w/r/t \( \langle \cdot, \cdot \rangle_b \)). Linear algebra then dictates that
any boolean function $f : \{0, 1\}^n \to \mathbb{R}$ can be written uniquely as a linear combination of the Fourier basis functions $\{\chi_S : S \subseteq [n]\}$, and this representation is given by

$$f = \sum_{S \subseteq [n]} \langle f, \chi_S \rangle_b \chi_S = \sum_{S \subseteq [n]} \mathbb{E}_x[f(x)\chi_S(x)] \chi_S.$$ 

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 with respect to the norms $\langle \cdot, \cdot \rangle_b$ and $\langle \cdot, \cdot \rangle$; that is, a rotation that preserves distances. For any boolean functions $f, g : \{0, 1\}^n \to \{0, 1\}$ we have

$$P_x[f(x) \neq g(x)] = \langle f - g, f - g \rangle_b = \langle \hat{f} - \hat{g}, \hat{f} - \hat{g} \rangle = \|\hat{f} - \hat{g}\|^2,$$

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.

### 24.3 Linearity

**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 $f(x + y)$.

3. Accept $f$ if $f(x + y) = f(x)f(y)$.

This algorithm was analyzed by [BLR93b] as follows.

**Theorem 24.6.** Let $f : \{0, 1\}^n \rightarrow \{-1, 1\}$. 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 $P = \mathbb{P}_{x,y}[f(x)f(y) = f(x + y)]$. Let $Z \in \{0, 1\}$ be the indicator variable for the event that $f(x + y) = f(x)f(y)$. We have $P = \mathbb{E}[Z]$. 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

$$P = \mathbb{E}[Z] = \frac{1}{2} + \frac{1}{2} \mathbb{E}_x[f(x) \mathbb{E}_y[f(y)f(x + y)]] = \frac{1}{2} + \frac{1}{2} \langle f, h \rangle_b \overset{(b)}{=} \frac{1}{2} + \frac{1}{2} \langle \hat{f}, \hat{h} \rangle,$$

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

$$P = \frac{1}{2} + \frac{1}{2} \sum_S \hat{f}_S^3 \overset{(d)}{\leq} \frac{1}{2} + \frac{1}{2} \max_S \hat{f}_S$$
(e) applies the fact that \( \|\hat{f}\|^2 = 1 \) for \( f : \{0, 1\} \to \{-1, 1\} \). Rearranging, we have

\[
\hat{f}_S \geq 2P - 1
\]

for some set \( S \subseteq [n] \). But then

\[
4P[f(x) \neq \chi_S] = \|f - \chi_S\|_b^2 = \|f\|_b + \|\chi_S\|_b - 2\langle f, \chi_S \rangle = 2 - 2\hat{f}_S \leq 4(1 - P),
\]

as desired.

If \( f \) is linear, then the linearity test succeeds one hundred percent of the time. But then the above theorem asserts there exists a basis function \( \chi_S \) that agrees with \( f \) one hundred percent of the time. That is:

**Corollary 24.7.** All linear functions \( f : \{0, 1\}^n \to \{-1, 1\} \) are of the form \( \chi_S \) for some set \( S \).

**Locally correcting for linearity.**

**Theorem 24.8.** Let \( f : \{0, 1\} \to \{-1, 1\} \) be \( \epsilon \)-close to a basis function \( \chi_S : \{0, 1\} \to \{-1, 1\} \). Given \( x \in \{0, 1\}^n \), consider the random value \( f(x)f(x + y) \in \{-1, 1\} \) where \( y \in \{0, 1\}^n \) is sampled uniformly at random. Then

\[
P_y[f(y)f(x + y) = \chi_S(x)] \geq 1 - 2\epsilon.
\]

**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 \( \chi_S(x) = \chi_S(x + y)\chi_S(y) \).

**A remark on convolutions.** A key component of the proof of theorem 24.6 is the identity \( \hat{h}_S = \hat{f}_S^2 \) for the function \( h(x) = E_y[f(y)f(x + y)] \). More generally, for two boolean formulas \( f, g : \{0, 1\} \to \mathbb{R} \), the convolution of \( f \) and \( g \), denoted \( f \ast g \), is the function defined by

\[
(f \ast g)(x) = E_y[f(x)g(x + y)].
\]

The following identity is called *Plancheral’s identity* and generalizes the calculations used in theorem 24.6.

**Lemma 24.9.** Let \( f, g : \{0, 1\} \to \mathbb{R} \). Then \( (\hat{f} \ast \hat{g})_S = \hat{f}_S \hat{g}_S \) for all \( S \subseteq [n] \).

**Exercise 24.3.** Prove lemma 24.9.
24. Dictators

A function $f : \{0,1\}^n \to \{-1,1\}$ is a dictator if it is one of the singleton basis functions,$^2$

$$f = \chi_i \text{ for some } i \in [n].$$

We want to test if a function $f : \{0,1\}^n \to \{-1,1\}$ is a dictator. We do so by combining two tests. First, clearly, any dictator function is linear, which gives our first test.

1. **Linearity test:** Sample $x,y \in \{0,1\}^n$ independently and reject if $f(x+y) \neq f(x)f(y)$.

The second test is new. Let $\Omega = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ be the set of triplets where not all coordinates are equal. Abusing notation, we write $\Omega^n$ to denote the triplets of vectors $x,y,z \in \{0,1\}^n$ such that for all $i$, $(x_i,y_i,z_i) \in \Omega$. To sample a uniformly random $(x,y,z) \in \Omega$, one can independently sample, for each $i \in [n]$, three coordinates $(x_i,y_i,z_i) \in \Omega$ uniformly at random. We write $(x,y,z) \sim \Omega^n$ when $(x,y,z) \in \Omega^n$ is sampled uniformly at random.

For any dictator function $f = \chi_i$, and $(x,y,z) \in \Omega^n$, we have $(f(x),f(y),f(z)) \in \Omega$. This motivates our second test:

2. **Not-all-equal (NAE) test:** Sample $x,y,z \sim \Omega^n$. Reject $f$ unless $(f(x),f(y),f(z)) \in \Omega$.

**Theorem 24.10.** Let $f : \{0,1\}^n \to \{-1,1\}$ be a boolean function. Suppose $f$ passes both the linearity and not-all-equals test with probability $1 - \epsilon$ for $\epsilon \leq .1$. Then there exists a coordinate $i \in [n]$ such that

$$\mathbb{P}_x[f(x) \neq \chi_i(x)] \leq \epsilon,$$

where $x \in \{0,1\}^n$ is sampled uniformly at random.

To prove theorem 24.10, we first require the following lemma analyzing the not-all-equal test.

**Lemma 24.11.** Let $f : \{0,1\}^n \to \{-1,1\}$. Let $(x,y,z) \sim \Omega^n$. Then

$$\mathbb{P}[(f(x),f(y),f(z)) \in \Omega] \leq \frac{7}{9} + \frac{2}{9} \sum_{i=1}^n \hat{f}_i^2.$$

$^2$For ease of notation, we write $\chi_i$ instead of $\chi_{\{i\}}$.  

339
We will prove this lemma below in section 24.4. First, let us use it to prove theorem 24.10.

**Proof of theorem 24.10.** By the NAE test, we have \( \sum i \hat{f}_i^2 \geq 1 - 4.5\epsilon \). By the linearity test, we have that \( \hat{f}_S \geq 1 - 2\epsilon \) for some \( S \). But this set \( S \) must be a singleton \( \{i\} \) because otherwise we have

\[
1 = \|\hat{f}\|^2 \geq 1 - 4.5\epsilon + (1 - 2\epsilon)^2 > 1,
\]

a contradiction. Thus \( \hat{f}_i \geq 1 - 2\epsilon \) for some \( i \). Then

\[
4 P_x[f(x) \neq \chi_i(x)] \overset{(a)}{=} \|f - \chi_i\|^2_b = \|\hat{f} - \hat{\chi_i}\|^2_b (\hat{f}_i - 1)^2 + 1 - \hat{f}_i^2 = 2 - 2\hat{f}_i \leq 4\epsilon,
\]

as desired. (a) is by lemma 24.4. (b) takes the Fourier transform. (c) uses the identity \( \|\hat{f}\|^2 = 1 \) for all \( f : \{0,1\}^n \rightarrow \{-1,1\} \).

The dictatorship test we have just developed requires 6 queries to \( f \): three for the linearity test, and three for the not-all-equals test. We can reduce this to three queries at the cost of increase the error rate with a simple trick, as follows.

**Theorem 24.12.** Let \( n \in \mathbb{N} \). There is a 3-query test for the family of dictators \( D = \{\chi_i | i \in [n]\} \) with the following guarantee. Given a function \( f : \{0,1\}^n \rightarrow \{0,1\} \):

1. If \( f \in D \), then the test always accepts \( f \).
2. If \( f \) is \( \epsilon \)-far from any dictator function and \( \epsilon \leq .2 \), then the test \( f \) with probability \( \geq \epsilon/4 \).

**Proof.** We choose either the linearity test or not-all-equals test, randomly selecting one of the two with equal probability. Clearly, if \( f \) is a dictator, then the test always passes. Otherwise, suppose \( f \) fails the test with probability \( \leq p \). Consider the test where we run both tests on \( f \); \( f \) fails this test with probability \( \leq 2p \). It follows that for \( p \leq .05 \), \( f \) is at most \( 4p \)-far from some dictator.

**Families of dictators.** We can extend the dictator test above to subfamilies of dictator functions as follows. For any set \( S \subset [n] \), let

\[
D_S = \{\chi_i : \{0,1\}^n \rightarrow \{-1,1\} | i \in S\}
\]

be the set of dictator functions for coordinates \( i \in S \). Suppose that given \( f : \{0,1\}^n \rightarrow \mathbb{R} \) 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 24.10.

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 24.13.** 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 \( f \not\in D_S \) 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 \( i \in \chi_i \). \( \square \)

**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 \( N_p(x) \) be the distribution of random strings where each bit \( x_i \) is flipped independently with probability \( p \). The random function \( N_p \) 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

\[
(T_p f)(x) = \mathbb{E}_{y \sim N_p(x)}[f(y)].
\]

**Lemma 24.14.** Let \( f : \{0,1\} \to \mathbb{R} \) be a boolean function. For \( S \subset [n] \), \( \left( T_p f \right)_S = (1 - 2p)^{|S|} \hat{f}_S \).

**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

\[
(T_p \chi_S)(x) = \mathbb{E}_{y \sim N_p(x)}[\chi_S(y)] = \prod_{i \in S} \mathbb{E}_{y \sim N_p(x)}[(-1)^{y_i}] = \prod_{i \in S} ((1 - p)(-1)^{x_i} - p(-1)^{x_i})
\]

\[
= \prod_{i \in S} (1 - 2p)(-1)^{x_i} = (1 - 2p)^{|S|} \chi_S(x),
\]

as desired. \( \square \)
Analysis of the not-all-equals test. Finally, let us analyze the not-all-equals test and prove lemma 24.11.

Lemma 24.11. 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

\[
\text{NAE}(a, b, c) = \frac{1}{8} \left( (a - b)^2 + (a - c)^2 + (b - c)^2 \right) = \frac{1}{8} \left( 2a^2 + 2b^2 + 2c^2 - 2ab - 2ac - 2bc \right) = \frac{3}{4} - \frac{1}{4} (ab + ac + bc).
\]

Thus

\[
P = E[\text{NAE}(f(x), f(y), f(z))] = \frac{3}{4} - \frac{1}{4} E[f(x)f(y) + f(y)f(z) + f(x)f(z)]
\]

\[
\overset{(a)}{=} \frac{3}{4} - \frac{3}{4} E[f(x)f(y)]
\]

where (a) is by symmetry of \( \Omega \). Consider \( E[f(x)f(y)] \). We have

\[
E[f(x)f(y)] \overset{(b)}{=} \mathbb{E}_{x,y \sim \mathcal{N}_{2/3}(x)}[f(x)f(y)] = \left\langle f, T_{2/3} f \right\rangle_b = \left\langle \hat{f}, \left( T_{2/3} \hat{f} \right) \right\rangle
\]

\[
= \sum_S (-1/3)^{|S|} \hat{f}_S^2 \geq -\frac{1}{3} \sum_i \hat{f}_i^2 - \frac{1}{27} \sum_{|S| \geq 3, \text{ odd}} \hat{f}_S^2
\]

\[
\overset{(c)}{=} -\frac{1}{3} \sum_i \hat{f}_i^2 - \frac{1}{27} \left( 1 - \sum_i \hat{f}_i^2 \right) = -\frac{1}{27} - \frac{8}{27} \sum_i \hat{f}_i^2.
\]

Here (b) observes that \( x \) is sampled uniformly from \( \{0, 1\}^n \), and conditional on \( x \), \( y \) is distributed as \( \mathcal{N}_{2/3}(x) \). (c) applies the unitary Fourier transform. (d) is by
lemma 24.14. (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. \( \square \)

24.5 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 24.1. We restate the claim for the reader’s convenience.

**Theorem 24.1.** Let \( L \subseteq \{0, 1\}^n \). Let \( p = 2^{2n} \). Then there is a randomized algorithm \( A_L : \{0, 1\}^n \times \{0, 1\}^p \to \{0, 1\} \) that, given oracle access to \( x \in \{0, 1\}^n \) and \( y \in \{0, 1\}^p \), has the following properties.

(a) \( A_L \) makes 3 queries to bits of \( x \) or \( y \).

(b) If \( x \in L \), then there exists \( y \in \{0, 1\}^p \) such that \( A_L(x, y) = 1 \) always.

(c) If \( x \notin L \), then for all \( y \in \{0, 1\}^p \), we have

\[
P[A_L(x, y) = 0] \geq 0.01 \min_{y \in L} \frac{\|x - y\|_0}{n}.
\]

**Proof.** We sketched the algorithm in section 24.1 and now we describe it 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,

\[
D_L = \{ \chi_w : \{0, 1\}^N \to \{0, 1\} \mid w \in L \}.
\]

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 \( \chi_x : \{0, 1\}^N \to \{-1, 1\} \), 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 24.13.

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 24.3. 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 \( w_j \) (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 24.3, 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 \( Z \),

\[
P[H(Y, Z) \neq \chi_w(Z)] \leq C_2\epsilon
\]

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

\[
P_x[j \neq w_j] \leq \sum_j P[H(Y, Z_j) \neq w_j] + P[H(Y, Z_j) \neq x_j] \leq C_2\epsilon + 2\epsilon.
\]

(a) is by the union bound. The first term in (b) is by (24.3) 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 \( w \in L \).
24.6 Additional notes and materials

This chapter is motivated by the PCP theorem and limited in scope to the techniques that lead to the universal tester. There are many other applications of property testing and Boolean analysis. See [ODo14] for a booklength treatment on these topics. These notes are based on chapters 1, 2, and 7 of [ODo14]. Property testing also extends beyond boolean functions. We recommend Prof. Grigorescu’s Spring 2021 class on *sublinear time algorithms* for more topics in this area.

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

24.7 Exercises

**Exercise 24.1.** Complete the proof of theorem 24.2 by verifying that the graph-CSP described above satisfies the claimed properties.

**Exercise 24.2.** Verify eqs. (24.1) and (24.2) on page 335.

**Exercise 24.3.** Prove lemma 24.9.
Chapter 25

Online algorithms

25.1 Caching

At a very abstract level, a computer consists of memory holding data and a CPU that computes the data. In one generic step, it loads a few bytes of data from memory into the CPU registers, does some calculations on the registers, and writes the result back into memory.

The memory is supported by a series of devices with different tradeoffs between memory size and latency. The biggest device is the hard disk, which can hold terabytes of nonvolatile memory but is slow to read and write. Next we have RAM, which holds gigabytes of volatile memory, and is much faster than the hard drive. Finally the CPU also has its own local ($L_1$ and $L_2$) cache, which holds at most a few megabytes of memory, but is extremely fast to access due to its proximity to the CPU.

At the end of the data, the memory is backed by the hard disk, and the RAM and local cache act as faster intermediate layers. When the CPU needs a piece of memory $x$, it first checks the local cache. If the local cache does not have $x$, then it checks the RAM. If $x$ is not in RAM, then finally the computer retrieves $x$ from disk. The RAM and the cache are both populated with $x$ along the way to the CPU. In the likely event that we read or write $x$ again, it will already be cached in faster memory.

Thus the effective speed of a computer is to large extent dominated by the frequency with which we have to hit the hard disk. This bottleneck can be addressed by with good cache strategies that try to maximize the odds of the data sitting in memory. For example, when $x$ was placed into memory above, we need to evict another item $y$
to make room for $x$. Different strategies from choosing $y$ can make a big difference in performance.

**The model.** We will analyze cache strategies in a simplified model. We assume there are $n$ pieces of data of identical size, and $k$ cache slots that can each hold one piece of data. In general, $n$ is much larger than $k$. The input is an online sequence of data requests, in the form of indices out of $[n]$. Each request $i$ is served from cache, so if $i$ is not already in one of the $k$ slots, then we have to choose which of the $k$ slots to place it in. The general goal is to minimize the number of cache misses.

A salient point is that the requests are made online: we have to choose which item to evict without any knowledge of future requests.

The only design decision is in the eviction policy: when the requested data $x$ is not in cache, which item $y$ should be evicted to make space for $x$? Here are a few approaches:

- **Least frequently used (LFU):** Evict the item $y$ that has been accessed the fewest number of times (since being put in cache).

- **Least recently used (LRU):** Evict the item $y$ with the oldest time of last access.

- **Not recently used (NRU), a.k.a. 1-bit LRU:** Whenever an item is accessed, mark it. Evict unmarked items. If all cache items are marked, then remove all marks and try again.

LRU can be seen as a special case of NRU.

**Competitive analysis.** We typically analyze algorithms from a worst-case point of view. For caching, the worst-case perspective would try to bound the total number of cache misses. In the online model, however, the worst-case is unbounded: the adversary can always request an item out of cache and force a cache miss.

So maybe there is no good cache strategy because the data requests are simply impossible to cache. But if there is a good cache strategy, then we can find a comparably good one? Suppose that for a fixed sequence of data requests $i_1, i_2, \ldots \in \mathbb{N}$, it was possible to have $m$ cache misses in hindsight. Is it possible for us, operating online, to also get $m$ misses? $100m$ misses? $mk$ misses? etc. In competitive analysis, we want to minimize the competitive ratio,

$$\text{competitive ratio} \overset{\text{def}}{=} \frac{\text{our misses}}{\text{OPT misses}}$$

where OPT is the minimum number of misses in hindsight.
Least frequently used (LFU). Least frequently used has an unbounded competitive ratio. We leave the proof to the reader as an exercise.

Exercise 25.1. Prove that LFU has unbounded competitive ratio. (That is, for all \( L > 0 \), give a sequence of requests for which LFU obtains a competitive ratio \( \geq L \).)

Not-recently-used (NRU) and least-recently-used (LRU). Next we analyze the not-recently-used and least-recently-used eviction strategies.

Theorem 25.1. NRU and LRU have competitive ratio at most \( k \).

Proof. LRU is a special case of NRU, so we only need to prove the bound for NRU. Consider a sequence of requests

\[ i_1, i_2, \ldots \in [n]. \]

We split the sequence whenever NRU resets all of its marks. We call the contiguous subsequence of requests between resets a run.

Each run has \( k \) distinct items. The first item after a run is distinct from the \( k \) distinct items in the run. Thus: any eviction strategy must have a miss on one of the last \( k - 1 \) distinct requests of the run, or miss on the first item after the run. That is, OPT makes at least 1 mistake per run.

On the other hand, NRU makes \( k \) mistakes per run. All put together, we have

\[
\frac{\text{# NRU misses}}{\text{# OPT misses}} \leq \frac{k(\text{# runs})}{\text{# runs}} = k,
\]

as desired. \( \square \)

Randomized NRU. Lastly we consider a randomized variation of the NRU algorithm. It starts from the NRU framework: marking items as they are accessed, evicting unmarked items, and unmarking all items when all items are marked. The key distinction is that when evicting an unmarked item, it chooses one uniformly at random.

Theorem 25.2. Randomized NRU has competitive ratio at most \( 2 \ln(k) \) in expectation.

Proof. As before, consider a sequence of requests

\[ i_1, i_2, \ldots \in [n], \]

and split the sequence into runs whenever all the marks are reset. There are \( k \) distinct items per run. Of these \( k \) items we distinguish two types:
1. “New” items that were not in the previous run.

2. “Repeat” items that were also in the previous run.

We define a potential function \( \Phi \) equal to the number of items that are in \( \text{OPT} \)'s cache, but not in the NRU cache.

Fix a single run. Let \( \Phi_{\text{in}} \) and \( \Phi_{\text{out}} \) be the value of \( \Phi \) at the beginning and the end of the run, respectively. Let \( \ell \) be the number of new items in the run; that leaves there are \( k - \ell \) repeat items in the run.

Consider \( \text{OPT} \). At the beginning of the run, out of \( \ell \) new pages not in the NRU cache, at most \( \Phi_{\text{in}} \) of them are in \( \text{OPT} \)'s cache. So \( \text{OPT} \) misses at least \( \ell - \Phi_{\text{in}} \) new pages.

Now consider \( \Phi_{\text{out}} \). \( \Phi_{\text{out}} \) is the number of items in \( \text{OPT} \)'s cache but not in the NRU cache at the end of the run. But the NRU cache is filled with the \( k \) items in the run. So \( \text{OPT} \) must have kicked out these \( \Phi_{\text{out}} \) items, and incurred \( \Phi_{\text{out}} \) cache missed, during the run.

So \( \text{OPT} \) has at least \( \ell - \Phi_{\text{in}} \) misses, and at least \( \Phi_{\text{out}} \) misses. Average these bounds together, \( \text{OPT} \) has at least

\[
\frac{\ell + \Phi_{\text{out}} - \Phi_{\text{in}}}{2}
\]

misses in the run.

Now consider NRU. The NRU cache has \( \ell \) cache misses for the \( \ell \) new items. Consider the repeat items.

1. For the first repeat item, there is at most an \( \ell/k \) change we evicted the first repeat item before it was requested.

2. For the second repeat item, there is at most an \((\ell + 1)/k\) chance we evicted the second repeat item before it was requested.

3. In general, for the \( h \)th repeat item, there is at most an \((\ell + h - 1)/k\) chance we evicted the \( h \)th repeat item before it was requested.

By linearity of expectation,

\[
\mathbb{E}[\# \text{ misses on repeats}] = \sum_{h=1}^{k-\ell} \mathbb{P}[\text{missing on the } h \text{th repeat item}]
\leq \sum_{h=1}^{k-\ell} \frac{\ell + h - 1}{k} \leq \sum_{h=1}^{k-\ell} \frac{\ell}{h + 1 - h} \leq \ell \ln(k).
\]

For (a) we observe that \((\ell + h - 1)(k + 1 - h) \leq \ell k\) by AM-GM.
25. Online algorithms

25.2. Buy-or-rent

To review, in a single run, OPT makes at least \(0.5(\ell + \Phi_{\text{out}} - \Phi_{\text{in}})\) mistakes, and randomized NRU makes at most \(\ell \ln(k)\) mistakes in expectation.

Let \(L\) denote the total number of new items over the entire sequence. Over all the requests, the number of OPT misses is at least \(L/2\) plus half of the total change in \(\Phi\). Initially \(\Phi = 0\) and at the end it is nonnegative so the total change is nonnegative. Thus OPT makes at least \(L/2\) mistakes.

Meanwhile the randomized NRU has \(L \ln(k)\) cache misses in expectation. Thus the expected competitive ratio is bounded by

\[
E[\text{competitive ratio}] \leq \frac{L \ln(k)}{L/2} = 2 \ln(k),
\]

as desired.

25.2 Buy-or-rent

Sometimes we have a nontrivial choice between buying and renting. For example those without a car can use Lyft or Uber to get around. But after a lot of taxi fares, buying a car becomes more appealing. Should you buy, or continue to rent? You don’t entirely know what the future holds.

Another example is with skis. The ski resort rents skis, but it’s not very cheap. You can alternatively buy skies, which is initially more expensive than renting, but could you save money in the long run if you ski a lot. Should you rent or buy skis? (IMO it’s much harder to project future ski use than car use.)

Decisions like this pop up all over a computer. For example, should you keep a hard drive spinning when not in use? It takes a while to spin up a hard drive from rest. On the other hand it takes energy to keep it spinning and ready. The right decision depends on future data access patterns that you don’t know.

For fun we will frame the problem in terms of skis. Suppose it costs \(k\) dollars to buy skis, and 1 dollar a day to rent skis. Let \(\ell\) be the total number of days ski, in hindsight. The tricky part is that we don’t know \(\ell\). Every day we decide to buy or rent skis, not knowing if we will ever ski again.

If we knew \(\ell\), the decision would be trivial:

1. If \(\ell \leq k\), then rent every day.
2. If \(\ell \geq k\), then buy on the first day.

Thus \(\text{OPT} = \min\{k, \ell\}\) in hindsight. Our goal is to develop an online strategy competitive with OPT.

As a warmup, we challenge the reader to think of a 2-competitive algorithm.
Exercise 25.2. Design and analyze an online, $(2 - o(1))$-competitive algorithm for the ski-rental problem on the preceding page.

In fact 2 is optimal for deterministic algorithms:

Exercise 25.3. Prove that $2 - o(1)$ is the optimal competitive ratio for any deterministic algorithm for the ski-rental problem on the previous page.

Half-skis. As a thought experiment, supposed we relaxed the rules so that you could but one ski at a time for $k/2$ dollars, and rent 1 ski at a time for 1/2 dollars. (You still need 2 skis each day). Note that the offline optimal strategy does not change: you should buy or rent (both) depending on if $\ell \geq k$ or not. Thus $\text{OPT} = \min\{k, \ell\}$. But here one can get a competitive ratio better than 2. Let us assume $k$ is divisible by 8 for simplicity.

1. On day $5k/8$, buy one ski.
2. On day $k$, buy a second ski.

For $\ell < 5k/8$, we are optimal since we are only renting. For $\ell = 5k/8$, we pay

$$\frac{5k}{8} + \frac{k}{2} = \frac{9k}{8} = \frac{9}{5} \text{OPT}.$$ 

For $\ell = k$, we pay

$$\frac{5k}{8} + \frac{13k}{8} + k = \frac{29}{16}k \leq \frac{29}{16} \text{OPT}.$$ 

For $\ell$ in between $5k/8$ and $k$, the competitive ratio is only better than at $5k/8$ or at $k$.

The takeaway is that operating “fractionally” – here we allow for “half” purchases – leads to a better ratio.

Liquid skis. Suppose we pushed the thought experiment further and allowed you to buy or rent arbitrary fractions of a pair of skis. For each day $i$, let $x_i$ denote the fractional skis bought on day $i$. Let $y_i$ be the fractional skis bought on day $i$. We require 1 total ski on each day $i$:

$$x_1 + \cdots + x_i + y_i \geq 1 \text{ for all } i \in [\ell].$$

Our goal is to minimize the total cost,

$$k(x_1 + \cdots + x_\ell) + y_1 + \cdots + y_\ell.$$
Observe that the optimum policy in hindsight is the same, hence \( \text{OPT} = \min \{k, \ell\} \).

Here we will analyze the following heuristic. Let \( \delta > 0 \) be a parameter TBD.

Suppose we commit to spending \( 1 + \delta \) units per day until we’ve completely bought the skis. Note that \( y_i = 1 - x_1 - \cdots - x_i \) so \( x_i \) determines \( y_i \). For day 1,

\[
1 + \delta = kx_1 + (1 - x_1) \implies x_1 = \frac{\delta}{k - 1}
\]

For day 2,

\[
1 + \delta = kx_2 + (1 - x_1 - x_2) \implies x_2 = \frac{1}{k - 1}(\delta + x_1).
\]

In general, on the \( i \)th day,

\[
1 + \delta = kx_i + (1 - x_1 - \cdots - x_i) \implies x_i = \frac{1}{k - 1}(\delta + x_1 + \cdots + x_{i-1}).
\]

For the first \( k \) days, \( \text{OPT} \) pays 1 dollar per day, while we pay \( \delta \). So for \( \ell \leq k \), we have a competitive ratio of \( 1 + \delta \). We now choose \( \delta \) to ensure we own a full set of skis after \( k \) days, which guarantees that the competitive ratio is \( 1 + \delta \) for all \( \ell \). We have

\[
1 = x_1 + \cdots + x_k = \frac{\delta}{k - 1} \left( 1 + \left( \frac{k}{k - 1} \right) + \left( \frac{k}{k - 1} \right)^2 + \cdots + \left( \frac{k}{k - 1} \right)^{k-1} \right)
\]

\[
= \delta \left( \left( \frac{k}{k - 1} \right)^k - \left( \frac{k}{k - 1} \right) \right).
\]

Rearranging, we have

\[
\delta = \frac{1}{\left( \frac{k}{k - 1} \right)^k - \left( \frac{k}{k - 1} \right)},
\]

and

\[
\lim_{k \to \infty} \delta = \frac{1}{e - 1} \approx .582.
\]

Thus the competitive ratio \( 1 + \delta \) converges to \( \frac{e}{e - 1} \approx 1.582 \).
Random skis. Our improved competitive ratio is artificial since of course we cannot buy skis in arbitrary fractions. We convert the continuous strategy above to a fractional one by randomized rounding. The high-level idea is that we commit to buying in one of the first $k$ days, and for $i \in [k]$, interpret $x_i$ as the probability of buying skis on day $i$. This can be implemented by as follows.

1. Pick $\alpha \in [0, 1]$ uniformly at random.
2. Simulate the fractional algorithm, giving values $x_1, x_2, \ldots \in [0, 1]$.
3. On day $i$, if $x_1 + \cdots + x_i \geq \alpha$, buy skis.

The expected cost from buying skis is

$$E[\text{cost buying}] = k \sum_{i=1}^\ell \mathbb{P}[(\text{buy on day } i)] = k(x_1 + \cdots + x_\ell).$$

The probability we rent on day $i$ is

$$\mathbb{P}[(\text{rent on day } i)] = \mathbb{P}[\alpha > x_1 + \cdots + x_i] = 1 - x_1 - \cdots - x_i = y_i.$$

Thus the expected cost from renting is

$$E[\text{cost renting}] = \sum_{i=1}^\ell \mathbb{P}[(\text{rent on day } i)] = y_1 + \cdots + y_\ell.$$

Thus the total expected cost is

$$E[\text{cost}] = E[\text{cost buying}] + E[\text{cost renting}] = k(x_1 + \cdots + x_\ell) + y_1 + \cdots + y_\ell,$$

the same cost as our (impractical) fractional solution! Thus we have the same competitive ratio in expectation:

$$E[\text{competitive ratio}] = 1 + \delta \xrightarrow{k \to \infty} \frac{e}{e - 1}.$$

In conclusion:

**Theorem 25.3.** There is randomized algorithm for the ski-rental problem with competitive ratio converging to $\frac{e}{e - 1}$ for large $k$. 

353
25.3 Additional notes and materials

The analysis of deterministic NRU/LRU is from [ST85a]. The randomized NRU algorithm is from [FKLMSY91]. Additional notes on caching can be found in [Blu00] and [MR95, Chapter 13]. The buy-or-rent algorithm is from [KMMO94].

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

25.4 Exercises

Exercise 25.1. Prove that LFU has unbounded competitive ratio. (That is, for all $L > 0$, give a sequence of requests for which LFU obtains a competitive ratio $\geq L$.)

Exercise 25.2. Design and analyze an online, $(2 - o(1))$-competitive algorithm for the ski-rental problem on page 350.

Exercise 25.3. Prove that $2 - o(1)$ is the optimal competitive ratio for any deterministic algorithm for the ski-rental problem on page 350.

Exercise 25.4. Let $\ell < k$. Prove that an LRU cache of size $k$ is $k/(k + 1 - \ell)$-competitive with any cache of size $\ell$.

That means, for example, that an LRU cache of size $k$ is 2-competitive with any cache of size $k/2$. Some people find this bound more compelling.

You should be able to prove this by a short modification of the argument of the $k$-competitive bound. It suffices to point out which part of the argument should change, and how.

Exercise 25.5. Suppose you are going to the graduate student social thingy on the 3rd floor of Lawson. You can get to the third floor by either an elevator or the stairs. The elevator takes 15 seconds (once you get in), while the stairs take 2 minutes. Your goal is to get up to the third floor as fast as possible before the donuts are all taken.

You press the button to go up for the elevator. You don’t know how long it will take to come down. Do you wait or take the stairs?

1. Suppose you knew how long the elevator would take to arrive. What is the optimal choice, based on this wait?
2. Now suppose you don’t know how long the elevator would take. Design a deterministic algorithm that is competitive with the optimal solution (where you know how long the elevator would take).\(^1\)

3. Suppose you have a quarter in your pocket, which lands heads or tails with equal probability. You can toss the coin once every 15 seconds. Design a randomized algorithm with a (slightly) better competitive ratio than the deterministic one from the previous question.\(^2\)

\(^1\)I believe a 15/8 competitive ratio is possible.

\(^2\)I actually don’t know what the best competitive ratio would be, and I’m interested to see what everyone comes up with.
Chapter 26

Sparsification

Generally speaking, sparsification refers to the idea of taking one complex object and replacing it with a smaller and simpler one that preserves certain salient properties. For example, geometric $\epsilon$-samples replace arbitrarily large point sets with small, reweighted point sets while approximately preserving the measure of every range.

This chapter introduces two more examples of sparsification. The first is graph sparsification, where we reduce the number of edges in an undirected graph while approximately preserving all cuts. The second is matrix sparsification, where we sparsify sums of positive semi-definite matrices while preserving all of its spectral properties. The latter generalizes the former via the Laplacian. Due to time constraints we will not prove the sparsification results, but instead focus on some of their applications.

26.1 Graph sparsification

Let $G = (V, E)$ be an undirected graph with $m$

**Theorem 26.1 ([BK15]).** Let $G = (V, E)$ be an undirected graph, $m$ edges, $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{\geq 0}$. Then there exists a subgraph $\tilde{G} = (V, E')$ (where $E' \subseteq E$) and weights $\tilde{w} : E' \to \mathbb{R}_{\geq 0}$ such that:

(a) $\tilde{G}$ has $|\tilde{E}| \leq O\left(\frac{n \log(n)}{\epsilon^2}\right)$ edges.

(b) preserves the weight of all cuts of $G$ up to a $(1 + \epsilon)$-factor. That is, for all $S \subset V$,

$$(1 - \epsilon)\tilde{w}(\delta(S)) \leq w(\delta(S)) \leq (1 + \epsilon)\tilde{w}(\delta(S))$$

Moreover, $(\tilde{G}, \tilde{w})$
Due to time constraints we will note prove theorem 26.1. Instead we will show something weaker that still sheds some insight into theorem 26.1: we will compute a sparse reweighted subgraph where the minimum cut is preserved up to a $(1 \pm \epsilon)$-factor.

Let $\lambda$ denote the weight of the minimum cut in $G$. Let $\tau = \frac{\epsilon^2 \lambda}{c \log n}$ for a sufficiently large constant $c > 0$. Consider the randomized weights $\tilde{w}$ where for each edge $e \in E$, independently, we have

$$
\tilde{w}(e) = \tau \cdot \begin{cases} 
\left\lceil \frac{w(e)}{\tau} \right\rceil \quad &\text{with probability } p_e \overset{\text{def}}{=} \frac{w(e)}{\tau} - \left\lfloor \frac{w(e)}{\tau} \right\rfloor \\
\left\lfloor \frac{w(e)}{\tau} \right\rfloor \tau \quad &\text{with probability } 1 - p_e.
\end{cases}
$$

Equivalently, $\tilde{w}(e)$ is defined by

$$
\tilde{w}(e) \in \left\{ \left\lceil \frac{w(e)}{\tau} \right\rceil \tau, \left\lfloor \frac{w(e)}{\tau} \right\rfloor \tau \right\} \quad \text{and} \quad \mathbb{E}[\tilde{w}(e)] = w(e).
$$

Consider a cut $\delta(S)$, where $S \subsetneq V$. The randomized weight of $\delta(S)$,

$$
\tilde{w}(\delta(S)) = \sum_{e \in \delta(S)} \tilde{w}(e),
$$

is a sum in independent random variables where the random part of each random variable varies by at most $\tau$. The expected value is

$$
\mathbb{E}[\tilde{w}(\delta(S))] = w(\delta(S)).
$$

Since $\tau \leq (\epsilon^2/c \log n)w(\delta(S))$ for a large constant $c$, by the multiplicative Chernoff bound, we have

$$
|\tilde{w}(\delta(S)) - w(\delta(S))| \leq \epsilon w(\delta(S))
$$

with high probability.

In particular, if $\delta(S)$ is the minimum cut, then $\tilde{w}(\delta(S)) \leq (1 + \epsilon)\lambda$ with high probability. Thus the minimum cut with respect to $\tilde{w}$ is at most $(1 + \epsilon)\lambda$ with high probability.

Next we want to show that the minimum weight is at least $(1 - \epsilon)\lambda$ with high probability. Call a cut $\delta(S)$ bad if

$$
\tilde{w}(\delta(S)) \leq (1 - \epsilon)\lambda,
$$

and good otherwise. While any individual cut is good w/h/p, we cannot simply take a union bound over all cuts because they are exponentially many cuts.

Recall that there are at most $\binom{n}{2}$ minimum cuts in the graph (chapter 4). An extension of the same argument shows that there are at most $n^{2t}$ cuts of weight at most $t\lambda$, for all $t \in \mathbb{N}$ (exercise 4.4).

Now, for $t \in \mathbb{N}$, let $Q_t$ be the family of edge cuts with weight in the range $[t\lambda, (t + 1)\lambda)$...
26.2 Spectral sparsification

Recall that a matrix $A$ is positive semi-definite (PSD) if it is symmetric ($\langle x, Ay \rangle = \langle y, Axy \rangle$ for all $x, y$) and $\langle x, Ax \rangle \geq 0$ for all $x$. For example, the Laplacian $L_e$ of an edge $e = \{u, v\}$, defined by $\langle x, L_e x \rangle = (x_u - x_v)^2$ for $x \in \mathbb{R}^V$, is positive semi-definite. So is the Laplacian $L = \sum_{e \in E} w(e) L_e$ of an undirected graph $G = (V, E)$ with edge weights $w \in \mathbb{R}_{>0}^E$.

Let $A, B$ be two PSD matrices. We write $A \preceq B$ if $\langle x, Ax \rangle \leq \langle x, Bx \rangle$ for all $x \in \mathbb{R}^V$. $\preceq$ defines a partial order on the family of PSD matrices (over a fixed vector space). We say that $B$ is a $(1 \pm \epsilon)$-approximation of $A$ if

$$(1 - \epsilon)B \preceq A \preceq (1 + \epsilon)B.$$ 

If $B$ is an $(1 \pm \epsilon)$-approximation of $A$ then many properties of $B$ are within a $(1 \pm \epsilon)$-factor of $A$. For examples, the eigenvalues of $B$ match those of $A$ up to a $(1 \pm \epsilon)$-factor:

Exercise 26.1. Suppose $B$ is an $(1 \pm \epsilon)$-approximation of $A$. Prove that for all $i$, if $\mu_{A,i}$ is the $i$th largest eigenvalue of $A$, and $\mu_{B,i}$ is the $i$th largest eigenvalue of $B$, then

$$(1 - \epsilon)\mu_{B,i} \leq \mu_{A,i} \leq (1 + \epsilon)\mu_{B,i}.$$ 

Let $A = \sum_{i=1}^m B_i$ be a sum of $m$ PSD matrices. We are interested in computing a sparse sum $\tilde{A} = \sum_{i=1}^m w_i B_i$, where $w_i \geq 0$ for all $i$, such that

(a) $(1 - \epsilon)\tilde{A} \preceq A \preceq (1 + \epsilon)\tilde{A}$.

(b) $w_i > 0$ for as few indices $i$ as possible.

Such an $\tilde{A}$ would be useful as it could replace $A$ in many applications with small loss, while being easier to compute with because it is smaller.

The matrix Chernoff bound

Theorem 26.2 (Matrix Chernoff bounds). Let $\epsilon \in (0, 1)$ and let $c > 0$ be sufficiently large. Let $\alpha = \epsilon^2/c \log(n)$ for $c > 0$ sufficiently large. Let $X_1, \ldots, X_n$ be independent and randomized positive semidefinite matrices with $X_i \preceq \alpha I$ for all $i$, and $E[X_1 + \cdots + X_n] = I$. Then

$$P[(1 - \epsilon)I \preceq X_1 + \cdots + X_n \preceq (1 + \epsilon)I] \geq 1 - n^{-\Omega(c)}.$$ 

358
The proofs of the matrix Chernoff bounds require technical tools beyond the scope of our course. Still let us try to offer some intuition.

Let $u \in \mathbb{R}^n$ be any unit vector. Consider the terms $\langle u, X_i u \rangle$, which (in a sense) project $X_i$ onto the direction $u$. The assumptions of the matrix Chernoff bound state that

$$0 \leq \langle u, X_i u \rangle \leq \frac{\epsilon^2}{c \log n}$$

for all $i$, and that

$$E \left[ \left\langle u, \left( \sum_{i=1}^n X_i \right) u \right\rangle \right] = E \left[ \sum_{i=1}^m \langle u, X_i u \rangle \right] = 1.$$

If we apply the standard Chernoff bound to the independent random variables $Y_i = \langle u, X_i u \rangle$, we get

$$1 - \epsilon \leq \sum_{i=1}^n \langle u, X_i u \rangle \leq 1 + \epsilon$$

with probability at least $1 - n^{-\Omega(c)}$.

Thus in any single direction, the random sum $\sum_i X_i$ is well concentrated. The strength of the matrix Chernoff bound is the assertion that the random sum $\sum_i X_i$ is simultaneously well-concentrated in all directions. This is much stronger than can be obtained by a union bounded argument over all directions...

**Sparsifying sums.**

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

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

**26.3 Exercises**

**Exercise 26.1.** Suppose $B$ is an $(1 \pm \epsilon)$-approximation of $A$. Prove that for all $i$, if $\mu_{A,i}$ is the $i$th largest eigenvalue of $A$, and $\mu_{B,i}$ is the $i$th largest eigenvalue of $B$, then

$$(1 - \epsilon)\mu_{B,i} \leq \mu_{A,i} \leq (1 + \epsilon)\mu_{B,i}.$$
Bibliography


362


[Nel16] Jelani Nelson. *Load balancing, k-wise independence, chaining, linear probing*. Recorded lecture. 2016. URL: https://www.youtube.com/watch?v=WqBc0ZCU4Uw & list=PLZSOwxxB0UP4rJgf5ayhHWgw7akUWsf&index=3


<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
<th>Authors</th>
<th>Publication Details</th>
</tr>
</thead>
</table>

369


Appendix A

Homework assignments

![Comic strip: Calvin and Hobbes discussing homework](image)
Homework 0

- Due 11:59PM on Wednesday, August 31.
- Please be aware of appendix C.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

We recommend exercises 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.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 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.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 4.2
2. Exercise 5.1
3. Exercise 6.3
4. Exercise 6.4 (This one is the trickiest, IMO.)
5. Exercise 7.5
Homework 3

- Due 11:59PM on Friday, October 21.
- Please be aware of appendix C.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 8.3
2. Exercise 8.4
3. Exercise 9.1
4. Exercise 10.4
5. Exercise 11.4
Homework 4

- Due 11:59PM on Friday, November 4.
- Please be aware of appendix C.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 12.2
2. Exercise 13.1
3. Exercise 14.1
4. Exercise 15.1
**Homework 5**

- Due 11:59PM on Friday, November 18.
- Please be aware of appendix C.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 16.2
2. Exercise 17.2
3. Exercise 18.2
4. Exercise 19.6
5. Exercise 19.7
Homework 6

- Due 11:59PM on Friday, December 9.
- Please be aware of appendix C.5 (in the syllabus) regarding homework policies.
- Before submitting, we encourage you to ask yourself, *Is this really the simplest way to solve the problem? Is this really the clearest way to explain the solution?*

1. Exercise 20.3
2. Exercise 21.6
3. Problem 4.8 in [Vad12]. (This one is the trickiest, IMO.)
4. Exercise 23.2.
5. Exercises 23.5 and 23.6 (combined as one problem with three parts.)
Appendix B

Aggregate statistics

B.1 Homework

Below is a histogram of homework scores. For the word problems we drop the \( \lceil n/5 \rceil \) lowest scores, where \( n \) is the number of word problems submitted (and not excused). The median is 91.1, the average is 90.4, and the standard deviation is 9.1 (rounding to the nearest .1). Below is a histogram of all the homework scores.

B.2 Midterm

25 students took the first midterm. Below is a histogram of all the scores (properly weighted, and scaled out of 100). The median is 60.9, the average is 61.7, and the standard deviation is 13.4.
B. Aggregate statistics

B.3 Final

25 students took the final exam. Below is a histogram of all the scores (properly weighted, and scaled out of 100). The median is 64.3, the average is 63.7, and the standard deviation is 13.0.
B. Aggregate statistics

B.4. Total

Recall from the syllabus that the total score is computed at the end of the semester as

\[ 0.3 \text{homework} + 0.3 \text{midterm} + 0.4 \text{final}. \]

Below is a histogram of the total scores for the 25 students who completed the course. The median is 69.4, the average is 71.1, and the standard deviation is 9.3.
Appendix C

CS588: Syllabus, Policies, and Procedures

Welcome to CS588, which is about randomized algorithms. Please see the schedule (page 1) for a tentative list of topics. The class will also be similar to the Fall 2020 course (cf. https://fundamentalalgorithms.com/randomized/f20).

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 notes are provided. That said, the most closely aligned textbook with our course is:

---

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.

Other books and monographs overlapping with the course include:


The following courses from other institutions also have lecture notes that you may find helpful.

• *Randomized algorithms*, taught by David Karger at MIT. http://courses.csail.mit.edu/6.856/current/
• *Randomized algorithms*, taught by Sariel Har-Peled at UIUC. https://sarielhp.org/teach/17/b/
• *Randomized algorithms*, taught by Anupam Gupta and Avrim Blum at CMU. http://www.cs.cmu.edu/~avrim/Randalgs11/index.html
• *Algorithms for Big Data*, taught by Chandra Chekuri at UIUC. https://courses.engr.illinois.edu/cs498abd/fa2020/.
C.2 Correspondence

The course website is

www.fundamentalalgorithms.com/randomized,

where this document is posted.

Email. Please reserve emails to the instructor for true emergencies. Please use class time or office hours to ask questions or express concerns.

Piazza. There is a Piazza for the course at the following address.

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
- 40% Final

We compute numerical scores based on the weighting above (as a fractional value between 0 and 1), and then we curve the grades.

C.4 Exams

The midterms and final are each one part multiple choice and one part word problems. We have reserved classrooms through the school to give students an extended period of time for the tests.

C.4.1 What’s on the midterm?

A few students have asked about midterm 1 and here’s what I recently wrote to one of them.

I have not made the test yet and in fact I’ve never made an exam for randomized algorithms before, so I cannot speak with much certainty.

That said, two broad goals are:
1. Help identify points of weakness in each student’s understanding of the material, so we can address them.

2. (Gently) incentivize the class to learn all the material, stay engaged, etc.

So I plan to try to cover everything we’ve covered, including the material this upcoming week (though perhaps slightly less emphasized). I expect to have a mix of multiple-choice or short-answer problems and a few longer word problems.

I will also prepare some kind of cheat sheet with all the inequalities we use, such as Markov’s inequality, Chernoff’s inequality, etc. I will send everyone this cheat sheet before the exam.

Here are some ways to practice and study:

- Reviewing the lectures (obviously).
- Making sure you understand the solutions to the homework, even if you didn’t get them right the first time.
- There are many more problems in the notes that I did not assign. While I do not have solutions written up for them, I can quickly tell you in person if you are on the right track for any of them. I also encourage students to discuss solutions with one another; perhaps over Piazza.
- I may have problems like “Prove Chebyshev’s inequality” or “Prove the multiplicative Chernoff bound (for any constant in the exponent).” So I would learn the proofs of some of these basic tools.
- For some of the recent lectures, it may be helpful to at least attempt the problems in homework 3, even if they are not due until after the midterm. The problems are meant to help you understand the material.
- More problems can be found in the Motwani-Raghavan book. While we have mostly covered different algorithms from that book, you can still find exercises about more general themes; e.g., conditional probability, linearity of expectation, concentration bounds, etc.
- I found a practice final from Sariel’s website here. [https://sarielhp.org/teach/17/b/sp14/sp14_final.pdf](https://sarielhp.org/teach/17/b/sp14/sp14_final.pdf)
C.5 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).

Here are some articles about writing:

- Terry Tao: https://terrytao.wordpress.com/advice-on-writing-papers/.
- Cormac McCarthy: https://www.nature.com/articles/d41586-019-02918-5. (Accessible via school library.)

**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-fifth of word problem scores will be dropped. More precisely, if there are \( n \) total word 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 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.6 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

In particular, we are all expected to uphold the Protect Purdue Pledge.

C.6.1 Quarantining

If you must quarantine or isolate at any point in time during the semester, please reach out to any of the staff (preferable 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
C. Syllabus, policies, and procedures

C.7. Academic integrity

Kent Quanrud
Fall 2022

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.7 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.8 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.9 Purdue’s Honor Pledge

As a boilemaker 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.
C.10 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.11 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.12 Students with Disabilities

Purdue University is required to respond to the needs of the students with disabilities as outlined in both the Rehabilitation Act of 1973 and the Americans with Disabilities Act of 1990 through the provision of auxiliary aids and services that allow a student with a disability to fully access and participate in the programs, services, and activities at Purdue University. If you have a disability that requires special academic accommodation, please make an appointment to speak with the instructor within the first three (3) weeks of the semester in order to discuss any adjustments.

It is the student’s responsibility to notify the Disability Resource Center (http://www.purdue.edu/drc) of an impairment/condition that may require accommodations and/or classroom modifications. We cannot arrange special accommodations without confirmation from the Disability Resource Center.

C.13 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 meant to stop class or research and immediately evacuate the building. Proceed to your Emergency Assembly Area away from building doors. Remain outside until police, fire, or other emergency response personnel provide additional guidance or tell you it is safe to leave. All Hazards Outdoor Emergency Warning sirens mean to immediately seek shelter (Shelter in Place) in a safe location within the closest building. “Shelter in place” means seeking immediate shelter inside a building or University residence. This course of action may need to be taken during a tornado, a civil disturbance including a shooting or release of hazardous materials in the outside air. Once safely inside, find out more details about the emergency. Remain in place until police, fire, or other emergency response personnel provide additional guidance or tell you it is safe to leave. In both cases, you should seek additional clarifying information by all means possible: Purdue Home page, email alert, TV, radio, etc. Review the Purdue Emergency Warning Notification System multi-communication layers at http://www.purdue.edu/ehps/emergencypreparedness/warning-system.html. Please review the Emergency Response Procedures at https://www.purdue.edu/emergencypreparedness/flipchart/index.html. Please review the evacuation routes, exit points, emergency assembly area and shelter in place procedures and locations for the building. Video resources include a 20-minute active shooter awareness video that illustrates what to look for and how to prepare and react to this type of incident. See http://www.purdue.edu/securepurdue/police/video/

C.14 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.15 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). 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). 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.16 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,
C. Syllabus, policies, and procedures

C.17 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.18 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.19 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.20 Changes to the syllabus

This syllabus is subject to change and changes will be announced appropriately.