Notes on Randomized Algorithms
CS 469/569: Spring 2011

James Aspnes

2011-11-10 18:43

Contents

Preface **x**

Syllabus **xi**

Lecture schedule **xiv**

1 Randomized algorithms **1**
1.1 A trivial example 2
1.2 A less trivial example: randomized QuickSort 2
1.2.1 Brute force method: solve the recurrence 3
1.2.2 Clever method: use linearity of expectation 4
1.3 Karger's min-cut algorithm 5
1.4 Classifying randomized algorithms by their goals 7
1.4.1 Las Vegas vs Monte Carlo 7
1.4.2 Randomized complexity classes 8
1.5 Classifying randomized algorithms by their methods 9

2 Probabilistic recurrences **11**
2.1 Recurrences with constant cost functions 11
2.2 Examples 11
2.3 The Karp-Upfal-Wigderson bound 12
2.3.1 Waiting for heads 14
2.3.2 Quickselect 14
2.3.3 Tossing coins 14
2.3.4 Coupon collector 15
2.3.5 Chutes and ladders 15
2.4 High-probability bounds 16
2.4.1 High-probability bounds from expectation bounds . . 17
2.4.2 Detailed analysis of the recurrence 17
2.5 More general recurrences 18

3 Probabilistic inequalities **19**
3.1 Basics . . . 19
3.1.1 Law of total probability . . . 19
3.1.2 Union bound (Boole's inequality) . . . 20
3.1.2.1 Examples . . . 20
3.1.3 Linearity of expectation . . . 20
3.1.4 Markov's inequality . . . 21
3.1.4.1 Examples . . . 21
3.1.5 Jensen's inequality . . . 22
3.1.5.1 Examples . . . 22
3.2 Concentration bounds . . . 23
3.2.1 Chebyshev's inequality . . . 23
3.2.1.1 Examples . . . 24
Flipping coins . . . 24
Flipping non-independent coins . . . 24
Balls in bins . . . 24
Lazy select . . . 24
3.2.2 Chernoff bounds . . . 26
3.2.2.1 The classic Chernoff bound . . . 26
3.2.2.2 Chernoff bound variants . . . 28
3.2.2.3 Lower bound version of Chernoff bounds . . . 29
3.2.2.4 Other tail bounds for the binomial distribution 30
3.2.2.5 Applications . . . 30
Flipping coins . . . 30
Balls in bins again . . . 30
Flipping coins, central behavior . . . 31
Valiant's randomized hypercube routing . . . 31
3.2.3 Azuma-Hoeffding inequality . . . 33
3.2.3.1 Hoeffding's inequality . . . 34
3.2.3.2 Azuma's inequality . . . 37
3.2.3.3 The method of bounded differences . . . 40
3.2.3.4 Applications . . . 41
3.3 Anti-concentration bounds . . . 44
3.3.1 The Berry-Esseen theorem . . . 44
3.3.2 The Littleword-Offord problem . . . 45

4 The probabilistic method **46**
4.1 Basic idea . . . 46
4.1.1 Unique hats . . . 47
4.1.2 Large cuts . . . 48

4.1.3 Ramsey numbers 49
4.1.4 Cycles in tournaments 50
4.2 Applications to MAX-SAT 50
4.3 The Lovász Local Lemma 54
4.3.1 General version 54
4.3.2 Symmetric version 55
4.3.3 Applications 56
4.3.3.1 Graph coloring 56
4.3.3.2 Satisfiability of k-CNF formulas 56
4.3.4 Non-constructive proof 57
4.3.5 Constructive proof 59

5 Derandomization **63**
5.1 Deterministic vs. randomized algorithms 64
5.2 Adleman's theorem 65
5.3 The method of conditional probabilities 66
5.3.1 A trivial example 67
5.3.2 Deterministic construction of Ramsey graphs 67
5.3.3 Set balancing 68

6 Martingales and stopping times **69**
6.1 The optional stopping theorem 69
6.2 Proof of the optional stopping theorem 70
6.3 Variants 71
6.4 Applications 72
6.4.1 Random walks 72
6.4.2 Wald's equation 74
6.4.3 Waiting times for patterns 74

7 Markov chains **76**
7.1 Basic definitions and properties 76
7.1.1 Examples 77
7.1.2 Classification of states 78
7.1.3 Reachability 79
7.2 Stationary distributions 80
7.2.1 The ergodic theorem 80
7.2.1.1 Proof 81
7.2.2 Reversible chains 82
7.2.2.1 Basic examples 83
7.2.2.2 Metropolis-Hastings 83

7.3 Bounding convergence rates using the coupling method 84
7.3.1 The basic coupling lemma 85
7.3.2 Random walk on a cycle 86
7.3.3 Random walk on a hypercube 87
7.3.4 Various shuffling algorithms 88
Move-to-top 88
Random exchange of arbitrary cards 89
Random exchange of adjacent cards 90
Real-world shuffling 91
7.3.5 Path coupling 91
7.3.5.1 Sampling graph colorings 92
7.3.5.2 Sampling independent sets 93
7.3.5.3 Metropolis-Hastings and simulated annealing 97
Single peak 97
Single peak with very small amounts of noise . 98
7.4 Spectral methods for reversible chains 99
7.4.1 Time-reversed chains 99
7.4.2 Spectral properties of a reversible chain 100
7.4.3 Conductance 103
7.4.4 Edge expansion using canonical paths 104
7.4.5 Congestion 106
7.4.6 Examples 107
7.4.6.1 Lazy random walk on a line 107
7.4.6.2 Random walk on a hypercube 107
7.4.6.3 Matchings in a graph 108
7.4.6.4 Perfect matchings in dense bipartite graphs . 110

8 Approximate counting 113
8.1 Exact counting 113
8.2 Counting by sampling 114
8.3 Approximating #DNF 115
8.4 Approximating #KNAPSACK 116

9 Hashing 119
9.1 Hash tables 119
9.2 Universal hash families 120
9.2.1 Example of a 2-universal hash family 122
9.3 FKS hashing 123
9.4 Cuckoo hashing 124
9.4.1 Structure 124

9.4.2 Analysis . . . 125
9.4.3 Practical issues . . . 127
9.5 Bloom filters . . . 128
9.5.1 False positives . . . 128
9.5.2 Applications . . . 130
9.5.3 Comparison to optimal space . . . 130
9.5.4 Counting Bloom filters . . . 131
9.5.5 Count-min sketches . . . 132
9.5.5.1 Initialization and updates . . . 132
9.5.5.2 Queries . . . 132
9.5.5.3 Finding heavy hitters . . . 135
9.6 Locality-sensitive hashing . . . 135
9.6.1 Approximate nearest neighbor search . . . 136
9.6.2 Locality-sensitive hash functions . . . 136
9.6.2.1 Constructing an (r_1, r_2)-PLEB . . . 137
9.6.2.2 Hash functions for Hamming distance . . . 138
9.6.2.3 Hash functions for ℓ_1 distance . . . 139

10 Randomized distributed algorithms 141
10.1 Randomized consensus . . . 142
10.1.1 Impossibility of deterministic algorithms . . . 142
10.1.2 Bounds on randomized algorithms . . . 143
10.2 Conciliators . . . 143
10.2.1 One-register lower bound . . . 145
10.2.2 An open problem . . . 147

A Assignments 149
A.1 Assignment 1: due Wednesday, 2011-01-26, at 17:00 . . . 149
A.1.1 Bureaucratic part . . . 149
A.1.2 Rolling a die . . . 149
A.1.3 Rolling many dice . . . 151
A.1.4 All must have candy . . . 151
A.2 Assignment 2: due Wednesday, 2011-02-09, at 17:00 . . . 152
A.2.1 Randomized dominating set . . . 152
A.2.2 Chernoff bounds with variable probabilities . . . 154
A.2.3 Long runs . . . 155
A.3 Assignment 3: due Wednesday, 2011-02-23, at 17:00 . . . 157
A.3.1 Longest common subsequence . . . 157
A.3.2 A strange error-correcting code . . . 159
A.3.3 A multiway cut . . . 160

A.4 Assignment 4: due Wednesday, 2011-03-23, at 17:00 161
A.4.1 Sometimes successful betting strategies are possible . 161
A.4.2 Random walk with reset 163
A.4.3 Yet another shuffling algorithm 164
A.5 Assignment 5: due Thursday, 2011-04-07, at 23:59 166
A.5.1 A reversible chain 166
A.5.2 Toggling bits 166
A.5.3 Spanning trees 168
A.6 Assignment 6: due Monday, 2011-04-25, at 17:00 170
A.6.1 Sparse satisfying assignments to DNFs 170
A.6.2 Detecting duplicates 170
A.6.3 Balanced Bloom filters 172
A.7 Final exam 174
A.7.1 Leader election 175
A.7.2 Two-coloring an even cycle 175
A.7.3 Finding the maximum 177
A.7.4 Random graph coloring 177

B Sample assignments from Spring 2009 semester 179
B.1 Final exam, Spring 2009 179
B.1.1 Randomized mergesort (20 points) 179
B.1.2 A search problem (20 points) 180
B.1.3 Support your local police (20 points) 181
B.1.4 Overloaded machines (20 points) 182

C Discrete probability theory 183
C.1 Probability spaces and events 183
C.1.1 Conditional probability and independence 184
C.1.2 Random variables 185
C.1.3 Expectation and moments 185
C.1.4 Conditional expectation 186

Bibliography 188

Index 197

List of Figures

3.1 Comparison of Chernoff bound variants 29

List of Tables

7.1 Markov chain parameters 79
7.2 Classification of Markov chain states 79

9.1 Hash table parameters 120

List of Algorithms

9.1 Insertion procedure for cuckoo hashing. Adapted from [Pag06]. 126
10.1 Impatient first-mover conciliator from [Asp10] 143
A.1 Dubious duplicate detector 171
A.2 Randomized max-finding algorithm 177

Preface

These are notes for the Spring 2011 semester version of the Yale course CPSC 469/569 Randomized Algorithms. This document also incorporates the lecture schedule and assignments, as well as some sample assignments from previous semesters. Because this is a work in progress, it will be updated frequently over the course of the semester.

The intent of this document is to update and replace the course notes previously scattered across a MoinMoin wiki at `http://pine.cs.yale.edu/pinewiki/`. These earlier notes can be found in the preserved web site for the Spring 2009 version of the course, at `http://pine.cs.yale.edu/pinewiki/CS469/2009/`.

Much of the structure of the course follows the textbook, Motwani and Raghavan's *Randomized Algorithms* [MR95], with some material from Mitzenmacher and Upfal's *Probability and Computing: Randomized Algorithms and Probabilistic Analysis* [MU05]. In most cases you'll find these textbooks contain much more detail than what is presented here, so it is probably better to consider this document a supplement to them than to treat it as your primary source of information.

Syllabus

Description

A study of randomized algorithms from several areas: graph algorithms, algorithms in algebra, approximate counting, probabilistically checkable proofs, and matrix algorithms. Topics include an introduction to tools from probability theory, including some inequalities such as Chernoff bounds.

Meeting times

Tuesdays and Thursday 1:00–2:15 in AKW 500.

On-line course information

Basic information about the course and links to other resources can be found at `http://pine.cs.yale.edu/pinewiki/CS469`. This site will also be used for announcements about the course.

The lecture schedule, course notes, and all assignments can be found in a single gigantic PDF file at `http://www.cs.yale.edu/homes/aspnes/classes/469/notes.pdf`. You should probably bookmark this file, as it will be updated frequently.

Office hours can be found in the course calendar at Google Calendar.

Textbook

The textbook for the class is: Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995. ISBN 0521474655. QA274 M68X 1995. Also available at `http://www.books24x7.com/marc.asp?isbn=0521474655` from Yale campus IP addresses.

Reserved books at E&AS library

These are other textbooks on randomized algorithms:

- Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. ISBN 0521835402. QA274 .M574X 2005. Also available at `http://www.books24x7.com/marc.asp?bookid=11418` from Yale campus IP addresses.

  This is a reasonably good introductory text on analysis of randomized algorithms with an emphasis on allocation problems.

- Juraj Hromkovič, *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms*. Springer, 2005. ISBN 9783540239499. QA274 .H76X 2005 (LC). Also available at `http://dx.doi.org/10.1007/3-540-27903-2` from Yale campus IP addresses.

  Intended to be a gentler introduction to randomized algorithms than Motwani and Raghavan, but not as comprehensive.

These are general references on probability theory:

- William Feller, *An Introduction to Probability Theory and Its Applications*, volumes 1 and 2. Wiley, 1968 (volume 1, 3rd edition); Wiley 1971 (volume 2, 2nd edition). QA273 F43 1968.

  The probability theory analog of Knuth's *Art of Computer Programming*: comprehensive, multiple volumes, every theoretical computer scientist of the right generation owns a copy. Volume 1, which covers discrete probability, is the most useful for computer science.

- Geoffrey R. Grimmett and David R. Stirzaker, *Probability and Random Processes*. Oxford University Press, 2001. ISBN 0198572220. QA273 G74X 2001.

  Similar in scope to Feller. A good alternative if you are on a budget.

Course requirements

Six homework assignments (60% of the semester grade) plus a final exam (40%).

Use of outside help

Students are free to discuss homework problems and course material with each other, and to consult with the instructor or a TA. Solutions handed in, however, should be the student's own work. If a student benefits substantially from hints or solutions received from fellow students or from outside sources, then the student should hand in their solution but acknowledge the outside sources, and we will apportion credit accordingly. Using outside resources in solving a problem is acceptable but plagiarism is not.

Clarifications for homework assignments

From time to time, ambiguities and errors may creep into homework assignments. Questions about the interpretation of homework assignments should be sent to the instructor at `aspnes@cs.yale.edu`. Clarifications will appear in an updated version of the assignment.

Late assignments

Late assignments will not be accepted without a Dean's Excuse.

Lecture schedule

As always, the future is uncertain, so you should take parts of the schedule that haven't happened yet with a grain of salt. Readings refer to chapters or sections in the course notes, except for those specified as in MR, which refer to the course textbook [MR95].

2011-01-11 Randomized algorithms. What they are and what we will do with them. Readings: Chapter 1; MR §§1.1–1.2, 1.5.

2011-01-13 Simple probabilistic recurrences. The Karp-Upfal-Wigderson bound [KUW88]. Readings: Chapter 2; MR §1.4.

2011-01-18 Probabilistic inequalities: basic random variable tools and concentration bounds. Readings: §§3.1–3.1.4, §3.2.1; MR §§3.1–3.3.

2011-01-20 Chernoff bounds and applications. Readings: §3.2.2; MR §§4.1–4.2.

2011-01-25 Hoeffding's inequality. Readings: §3.2.3.1.

2011-01-27 Azuma's inequality for martingales and its generalization to supermartingales. Using σ-algebras to represent knowledge. Readings: §3.2.3.2, §C.1.4; MR first part of §4.4.

2011-02-01 The method of bounded differences and other applications of Asuma's inequality. Readings: §3.2.3.3, §3.2.3.4; MR rest of §4.4.

2011-02-03 The probabilistic method: basic examples and randomized rounding. Readings: §§4.1 and 4.2; MR §§5.1 and 5.2.

2011-02-08 The Lovász Local Lemma. Non-constructive version and applications. Readings: §4.3 through §4.3.4; MR §5.5

2011-02-10 Constructive proof of the Lovász Local Lemma. Readings: §4.3.5; [MT10] (preprint available at `http://arxiv.org/abs/0903.0544`).

2011-02-15 Derandomization. Readings: Chapter 5. MR §2.3, §5.6.

2011-02-17 Martingales and stopping times. Readings: Chapter 6; [MU05, Chapter 12].

2011-02-22 Markov chains: basic definitions and convergence properties. Readings: §§7.1 and 7.2, MR §§6.1 and 6.2.

2011-02-24 Proving Markov chain convergence using the coupling method. Readings: §7.3; [MU05, Chapter 11].

2011-03-01 More examples of coupling (one of which worked, one of which didn't, but should have). Stationary distributions of reversible chains. Basic idea of path coupling. Readings: §7.3.5.

2011-03-03 Reversible chains and some examples of path coupling that actually work. Readings: §7.2.2; [MU05, §11.6].

2011-03-22 More Markov chain convergence: spectral methods, conductance, and canonical paths. Readings: §§7.4.1–7.4.5 and §7.4.6.1.

2011-03-24 Mostly talking about Assignment 4 (see Appendix A.4), but also a bit about using canonical paths to bound mixing time on a hypercube. Readings: §7.4.6.2.

2011-03-29 Approximate counting: counting classes, approximating #DNF. Readings: §§8.1–8.3; MR §§11.1–11.2.

2011-03-31 More approximate counting: #KNAPSACK. Readings: §8.4.

2011-04-05 Hash tables: universal hash functions, perfect hashing, and FKS. Readings: §§9.1–9.3; MR §§8.4–8.5.

2011-04-07 More hash tables: Cuckoo hashing. Readings: §9.4; [PR04]

2011-04-12 Still more hashing: Bloom filters and variants. §9.5; [MU05, §5.5.3].

2011-04-14 Count-min sketches and data stream computation. Readings: §9.5.5; [MU05, §13.4].

2011-04-19 Locality-sensitive hashing. Readings: §9.6; [IM98].

2011-04-21 Randomed distributed algorithms. Readings: Chapter 10.

2011-05-04 Final exam, starting at 14:00, in WLH 114. It was a closed-book test covering all material discussed during the semester. See Appendix A.7

Chapter 1

Randomized algorithms

A randomized algorithm flips coins during its execution to determine what to do next. When considering a randomized algorithm, we usually care about its **expected worst-case** performance, which is the average amount of time it takes on the worst input of a given size. This average is computed over all the possible outcomes of the coin flips during the execution of the algorithm. We may also ask for a **high-probability bound**, showing that the algorithm doesn't consume too much resources most of the time.

In studying randomized algorithms, we consider pretty much the same issues as for deterministic algorithms: how to design a good randomized algorithm, and how to prove that it works within given time or error bounds. The main difference is that it is often easier to design a randomized algorithm—randomness turns out to be a good substitute for cleverness more often than one might expect—but harder to analyze it. So much of what one does is develop good techniques for analyzing the often very complex random processes that arise in the execution of an algorithm. Fortunately, in doing so we can often use techniques already developed by probabilists and statisticians for analyzing less overtly algorithmic processes.

Formally, we think of a randomized algorithm as a machine M that computes $M(x, r)$, where x is the problem input and r is the sequence of random bits. Because the running time may depend on the random bits, it is now a **random variable**—a function on points in some probability space. The probability space Ω consists of all possible sequences r, each of which is assigned a probability $\Pr[r]$ (typically $2^{-|r|}$), and the running time for M on some input x is generally given as an **expected value**[1] $\mathrm{E}_r(\mathrm{time}(M(x, r))$,

[1] See Appendix C for more details on this and other concepts from discrete probability theory.

where for any X,

$$\mathrm{E}_r[X] = \sum_{r \in \Omega} X(r) \Pr[r]. \tag{1.0.1}$$

We can now quote the performance of M in terms of this expected value: where we would say that a deterministic algorithms runs in time $O(f(n))$, where $n = |x|$ is the size of the input, we instead say that our randomized algorithm runs in **expected time** $O(f(n))$, which means that $\mathrm{E}_r[\mathrm{time}(M(x, r))] = O(f(|x|))$ for all inputs x.

This is distinct from traditional **worst-case analysis**, where there is no r and no expectation, and **average-case analysis**, where there is again no r and the value reported is not a maximum but an expectation over some distribution on x. The following trivial example shows the distinction.

1.1 A trivial example

Suppose we have two doors. Behind one door is a valuable prize, behind the other is nothing. Our goal is to obtain the prize after opening the fewest possible doors.

A deterministic algorithm tries one door, then the next. In the worst case, two doors are opened. In the average case, if we assume that both doors are equally likely to hide the prize, we open one door half the time and the other door half the time, or 3/2 doors on average. We can obtain the same expected time even in the worst case by flipping a coin ourselves to decide which door to open first. This gives a randomized algorithm, and because *we* flip the coin (instead of nature, in the case of the average-case algorithm), we can guarantee the good expected performance no matter what the person hiding the prize does.

We've already used one mildly sophisticated techniques to analyze this algorithm: implicitly, we partitioned the probability space into two subsets, in one of which we opened the right door first and in one of which we opened the wrong door first.

1.2 A less trivial example: randomized QuickSort

The **QuickSort** algorithm [Hoa61a] works as follows. For simplicity, we assume that no two elements of the array being sorted are equal.

- If the array has > 1 elements,

  - Pick a **pivot** p uniformly at random from the elements of the array.
  - Split the array into A_1 and A_2, where A_1 contains all elements $< p$ elements $> p$.
  - Sort A_1 and A_2 recursively and return the sequence A_1, p, A_2.
- Otherwise return the array.

The splitting step takes exactly $n - 1$ comparisons, since we have to check each non-pivot against the pivot. We assume all other costs are dominated by the cost of comparisons. How many comparisons does randomized QuickSort do on average?

There are two ways to solve this problem: the dumb way and the smart way. We'll show both.

1.2.1 Brute force method: solve the recurrence

Let $T(n)$ be the expected number of comparisons done on an array of n elements. We have $T(0) = T(1) = 0$ and for larger n,

$$T(n) = \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-1-k)\right). \tag{1.2.1}$$

Why? Because there are n equally-likely choices for our pivot (hence the $1/n$), and for each choice the expected cost is $T(k) + T(n-1-k)$, where k is the number of elements that land in A_1. Formally, we are using here the **law of total probability**, which says that for any random variable X and partition of the probability space into events $B_1 \ldots B_n$, then

$$\mathrm{E}[X] = \sum B_i \,\mathrm{E}[X|B_i],$$

where

$$\mathrm{E}[X|B_i] = \frac{1}{\Pr[B_i]}\sum_{\omega \in B_i} X(\omega)$$

is the **conditional expectation** of X conditioned on B_i, which we can think of as just the average value of X if we know that B_i occurred. (See Section 3.1.1 for more details.)

So now we just have to solve this ugly recurrence. We can reasonably guess that when $n \geq 1, T(n) \leq an \log n$ for some constant a. Clearly this holds for $n = 1$. Now apply induction on larger n to get

$$\begin{aligned}
T(n) &= (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}(T(k) + T(n-1-k)) \\
&= (n-1) + \frac{2}{n}\sum_{k=0}^{n-1} T(k) \\
&= (n-1) + \frac{2}{n}\sum_{k=1}^{n-1} T(k) \\
&\leq (n-1) + \frac{2}{n}\sum_{k=1}^{n-1} ak \log k \\
&\leq (n-1) + \frac{2}{n}\int_{k=1}^{n} ak \log k \\
&= (n-1) + \frac{2a}{n}\left(\frac{n^2 \log n}{2} - \frac{n^2}{4} + \frac{1}{4}\right) \\
&= (n-1) + \frac{2a}{n}\left(\frac{n^2 \log n}{2} - \frac{n^2}{4} + \frac{1}{4}\right) \\
&= (n-1) + an \log n - \frac{an}{2} + \frac{a}{2n}.
\end{aligned}$$

If we squint carefully at this recurrence for a while we notice that setting $a = 2$ makes this less than or equal to $an \log n$, since the remaining terms become $(n-1) - n + 1/n = 1/n - 1$, which is negative for $n \geq 1$. We can thus confidently conclude that $T(n) \leq 2n \log n$ (for $n \geq 1$).

1.2.2 Clever method: use linearity of expectation

Imagine we use the following method for choosing pivots: we generate a random permutation of all the elements in the array, and when asked to sort some subarray A', we use as pivot the first element of A' that appears in our list. Since each element is equally likely to be first, this is equivalent to the actual algorithm. Pretend that we are always sorting the numbers $1 \ldots n$ and define for each pair of elements $i < j$ the **indicator variable** X_{ij} to be 1 if i is compared to j at some point during the execution of the algorithm and 0 otherwise. Amazingly, we can actually compute the probability of this event (and thus $\mathrm{E}[X_{ij}]$): the only time i and j are compared is if one

of them is chosen as a pivot before they are split up into different arrays. How do they get split up into different arrays? If some intermediate element k is chosen as pivot first, i.e., if some k with $i < k < j$ appears in the permutation before both i and j. Occurrences of other elements don't affect the outcome, so we can concentrate on the restriction of the permutations to just the numbers i through j, and we win if this restricted permutation starts with either i or j. This event occurs with probability $2/(j-i+1)$, so we have $\mathrm{E}[X_{ij}] = 2/(j-i+1)$. Summing over all pairs $i < j$ gives:

$$\begin{aligned}
\mathrm{E}\left[\sum_{i<j} X_{ij}\right] &= \sum_{i<j} \mathrm{E}[X_{ij}] \\
&= \sum_{i<j} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n-1} \frac{2}{k} \\
&= 2(n-1)H(n-1).
\end{aligned}$$

The first step here uses **linearity of expectation**: the fact that $\mathrm{E}[\sum X_i] = \sum \mathrm{E}[X_i]$ for any random variables X_i, even if they depend on each other in complicated ways. The rest is just algebra.

This is pretty close to the bound of $2n \log n$ we computed the dumb way. Note that in principle we can compute an *exact* solution by being more careful about the sum.

Which way is better? Solving the recurrence requires less probabilistic **handwaving** (a more polite term might be "insight") but more grinding out inequalities, which is a pretty common trade-off. Since I am personally not very clever I would try the brute-force approach first. But it's worth knowing about better methods so you can try them in other situations.

1.3 Karger's min-cut algorithm

Suppose we have a **multigraph**[2] G, and we want to partition the vertices into nonempty sets S and T such that the number of edges with one endpoint

[2]Unlike ordinary graphs, multigraphs can have more than one edge between two vertices.

in S and one endpoint in T is as small as possible (this is known as finding a **min-cut** or **minimum cut**, since the partition is a **cut** of the graph that has minimum total size). There are many sophisticated ways to do this. There is also a randomized algorithm due to David Karger [Kar93] that solves the problem using almost no sophistication at all (at least in the algorithm itself).

The main idea is that given an edge uv, we can construct a new multigraph G_1 by **contracting** the edge: in G_1, u and v are replaced by a single vertex, and any edge that used to have either vertex as an endpoint now goes to the combined vertex (edges with both endpoints in $\{u, v\}$ are deleted). Karger's algorithm is to contract edges chosen uniformly at random until only two vertices remain. All the vertices that got packed into one of these become S, the others become T. It turns out that this finds a minimum cut with probability at least $1/\binom{n}{2}$.

Theorem 1.3.1. *Given any min cut (S, T) of a graph G on n vertices, Karger's algorithm outputs (S, T) with probability at least $1/\binom{n}{2}$.*

Proof. Let (S, T) be a min cut of size k. Then the degree of each vertex v is at least k (otherwise $(v, G - v)$ would be a smaller cut), and G contains at least $kn/2$ edges. The probability that we contract an S–T edge is thus at most $k/(kn/2) = 2/n$, and the probability that we don't contract one is $1 - 2/n = (n - 2)/n$. Assuming we missed collapsing (S, T) the first time, we now have a new graph G_1 with $n-1$ vertices in which the min cut is still of size k. So now the chance that we miss (S, T) is $(n-3)/(n-1)$. We stop when we have two vertices left, so the last step succeeds with probability $1/3$. Multiplying all the probabilities together gives

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

□

If the graph has more than one min cut, this only makes our life easier. Note that since each min cut turns up with probability at least $1/\binom{n}{2}$, there can't be more than $\binom{n}{2}$ of them. But even if there is only one, we have a good chance of finding it if we simply re-run the algorithm substantially more than n^2 times.

1.4 Classifying randomized algorithms by their goals

1.4.1 Las Vegas vs Monte Carlo

One difference between QuickSort and Karger's min-cut algorithm is that QuickSort always succeeds (but may run for longer than you expect) while Karger's algorithm always runs in the same amount of time (but may fail to output a min cut, even if run multiple times). These are examples of two classes of randomized algorithms, which were originally named by Lśzl'o Babai [Bab79]:

- A **Las Vegas algorithm** fails with some probability, but we can tell when it fails. In particular, we can run it again until it succeeds, which means that we can eventually succeed with probability 1 (but with a potentially unbounded running time). Alternatively, we can think of a Las Vegas algorithm as an algorithm that runs for an unpredictable amount of time but always succeeds (we can convert such an algorithm back into one that runs in bounded time by declaring that it fails if it runs too long—a condition we can detect). QuickSort is an example of a Las Vegas algorithm.

- A **Monte Carlo algorithm** fails with some probability, but we can't tell when it fails. If the algorithm produces a yes/no answer and the failure probability is significantly less than 1/2, we can reduce the probability of failure by running it many times and taking a majority of the answers. Karger's min-cut algorithm in an example of a Monte Carlo algorithm.

The heuristic for remembering which class is which is that the names were chosen to appeal to English speakers: in Las Vegas, the dealer can tell you whether you've won or lost, but in Monte Carlo, *le croupier ne parle que Français*, so you have no idea what he's saying.

Generally, we prefer Las Vegas algorithms, because we like knowing when we have succeeded. But sometimes we have to settle for Monte Carlo algorithms, which can still be useful if we can get the probability of failure small enough. For example, any time we try to estimate an average by **sampling** (say, inputs to a function we are trying to integrate or political views of voters we are trying to win over) we are running a Monte Carlo algorithm: there is always some possibility that our sample is badly non-representative, but we can't tell if we got a bad sample unless we already know the answer we are looking for.

1.4.2 Randomized complexity classes

Las Vegas vs Monte Carlo is the typical distinction made by algorithm designers, but complexity theorists more elaborate classifications. These include algorithms with "one-sided" failure properties. For these algorithms, we never get a bogus "yes" answer but may get a bogus "no" answer (or vice versa). This gives us several complexity classes that act like randomized versions of **NP**, co-**NP**, etc.:

- The class **R** or **RP** (randomized **P**) consists of all languages L for which a polynomial-time Turing machine M exists such that if $x \in L$, then $Pr[M(x,r) = 1] \geq 1/2$ and if $x \notin L$, then $\Pr[M(x,r) = 1] = 0$. In other words, we can find a witness that $x \in L$ with constant probability. This is the randomized analog of **NP** (but it's much more practical, since with **NP** the probability of finding a winning witness may be exponentially small).

- The class co-**R** consists of all languages L for which a poly-time Turing machine M exists such that if $x \notin L$, then $\Pr[M(x,r) = 1] \geq 1/2$ and if $x \in L$, then $\Pr[M(x,r) = 1] = 0$. This is the randomized analog of co-**NP**.

- The class **ZPP** (zero-error probabilistic P) is defined as **RP** $\cap$ co-**RP**. If we run both our **RP** and co-**RP** machines for polynomial time, we learn the correct classification of x with probability at least $1/2$. The rest of the time we learn only that we've failed (because both machines return 0, telling us nothing). This is the class of (polynomial-time) Las Vegas algorithms. The reason it is called "zero-error" is that we can equivalently define it as the problems solvable by machines that always output the correct answer eventually, but only run in *expected* polynomial time.

- The class **BPP** (bounded-error probabilistic **P**) consists of all languages L for which a poly-time Turing machine exists such that if $x \notin L$, then $\Pr[M(x,r) = 1] \leq 1/3$, and if $x \in L$, then $\Pr[M(x,r) = 1] \geq 2/3$. These are the (polynomial-time) Monte Carlo algorithms: if our machine answers 0 or 1, we can guess whether $x \in L$ or not, but we can't be sure.

- The class **PP** (probabilistic **P**) consists of all languages L for which a poly-time Turing machine exists such that if $x \notin L$, then $\Pr[M(x,r) = 1] \geq 1/2$, and if $x \in L$, then $\Pr[M(x,r) = 1] < 1/2$. Since there is

only an exponentially small gap between the two probabilities, such algorithms are not really useful in practice; **PP** is mostly of interest to complexity theorists.

Assuming we have a source of random bits, any algorithm in **RP**, co-**RP**, **ZPP**, or **BPP** is good enough for practical use. We can usually even get away with using a pseudorandom number generator, and there are good reasons to suspect that in fact every one of these classes is equal to **P**.

1.5 Classifying randomized algorithms by their methods

We can also classify randomized algorithms by how they use their randomness to solve a problem. Some very broad categories:[3]

- **Avoiding worst-case inputs**, by hiding the details of the algorithm from the adversary. Typically we assume that an adversary supplies our input. If the adversary can see what our algorithm is going to do (for example, he knows which door we will open first), he can use this information against us. By using randomness, we can replace our predictable deterministic algorithm by what is effectively a random choice of many different deterministic algorithms. Since the adversary doesn't know which algorithm we are using, he can't (we hope) pick an input that is bad for all of them.

- **Sampling**. Here we use randomness to find an example or examples of objects that are likely to be typical of the population they are drawn from, either to estimate some average value (pretty much the basis of all of statistics) or because a typical element is useful in our algorithm (for example, when picking the pivot in QuickSort). Randomization means that the adversary can't direct us to non-representative samples.

- **Hashing**. Hashing is the process of assigning a large object x a small name $h(x)$ by feeding it to a **hash function** h. Because the names are small, the Pigeonhole Principle implies that many large objects hash to the same name (a **collision**). If we have few objects that we actually care about, we can avoid collisions by choosing a hash function that happens to map them to different places. Randomization helps here

[3] These are largely adapted from the introduction to [MR95].

by keeping the adversary from choosing the objects after seeing what our hash function is.

Hashing techniques are used both in **load balancing** (e.g., insuring that most cells in a **hash table** hold only a few objects) and in **fingerprinting** (e.g, using a **cryptographic hash function** to record a **fingerprint** of a file, so that we can detect when it has been modified).

- **Building random structures**. The **probabilistic method** shows the existence of structures with some desired property (often graphs with interesting properties, but there are other places where it can be used) by showing that a randomly-generated structure in some class has a nonzero probability of having the property we want. If we can beef the probability up to something substantial, we get a randomized algorithm for generating these structures.

- **Symmetry breaking**. In **distributed algorithms** involving multiple processes, progress may be stymied by all the processes trying to do the same thing at the same time (this is an obstacle, for example, in **leader election**, where we want only one process to declare itself the leader). Randomization can break these deadlocks.

Chapter 2

Probabilistic recurrences

Randomized algorithms often produce recurrences with ugly sums embedded inside them (see, for example, (1.2.1)). We'd like to have tools for pulling at least asymptotic bounds out of these recurrences without having to deal with the sums. This isn't always possible, but for certain simple recurrences we can make it work.

2.1 Recurrences with constant cost functions

Let us consider probabilistic recurrences of the form $T(n) = 1 + T(n - X_n)$, where X_n is a random variable with $0 < X_n \leq n$ and $T(0) = 0$. We assume we can compute a lower bound on $\mathrm{E}[X_n]$ for each n, and we want to translate this lower bound into an upper bound on $\mathrm{E}[T(n)]$.

2.2 Examples

- How long does it take to get our first heads if we repeatedly flip a coin that comes up heads with probability p? Even though we probably already know the answer to this, We can solve it by solving the recurrence $T(1) = 1 + T(1 - X_1), T(0) = 0$, where $\mathrm{E}[X_1] = p$.
- **Hoare's FIND** [Hoa61b], often called **QuickSelect**, is an algorithm for finding the k-th smallest element of an unsorted array that works like QuickSort, only after partitioning the array around a random pivot we throw away the part that doesn't contain our target and recurse only on the surviving piece. How many rounds of this must we do? Here $\mathrm{E}[X_n]$ is more complicated, since after splitting our array of size

n into piles of size n' and $n - n' - 1$, we have to pick one or the other (or possibly just the pivot alone) based on the value of k.

- Suppose we start with n biased coins that each come up heads with probability p. In each round, we flip all the coins and throw away the ones that come up tails. How many rounds does it take to get rid of all of the coins? (This essentially tells us how tall a skip list [Pug90] can get.) Here we have $\mathrm{E}[X_n] = (1-p)n$.

- In the **coupon collector problem**, we sample from $1 \ldots n$ with replacement until we see ever value at least once. We can model this by a recurrence in which $T(k)$ is the time to get all the coupons given there are k left that we haven't seen. Here X_n is 1 with probability k/n and 0 with probability $(n-k)/n$, giving $\mathrm{E}[X_n] = k/n$.

- Let's play **Chutes and Ladders** without the chutes and ladders. We start at location n, and whenever it's our turn, we roll a fair six-sided die X and move to $n - X$ unless this value is negative, in which case we stay put until the next turn. How many turns does it take to get to 0?

2.3 The Karp-Upfal-Wigderson bound

This is a bound on the expected number of rounds to finish a process where we start with a problem instance of size n, and after one round of work we get a new problem instance of size $n - X_n$, where X_n is a random variable whose distribution depends on n. It was original described in a paper by Karp, Upfal, and Wigderson on analyzing parallel search algorithms [KUW88]. The bound applies when $\mathrm{E}[X_n]$ is bounded below by a non-decreasing function $\mu(n)$.

Lemma 2.3.1. *Let a be a constant, let $T(n) = 1 + T(n - X_n)$, where for each n, X_n is an integer-valued random variable satisfying $0 \le X_n \le n - a$ and let $T(a) = 0$. Let $\mathrm{E}[X_n] \ge \mu(n)$ for all $n > a$, where μ is a positive non-decreasing function of n. Then*

$$\mathrm{E}[T(n)] \le \int_a^n \frac{1}{\mu(t)}\, dt. \tag{2.3.1}$$

To get an intuition for why this works, imagine that X_n is the speed at which we drop from n, expressed in units per round. Traveling at this speed, it takes $1/X_n$ rounds to cross from $k+1$ to k for any such interval

we pass. From the point of view of the interval $[k, k+1]$, we don't know which n we are going to start from before we cross it, but we do know that for any $n \geq k+1$ we start from, our speed will be at least $\mu(n) \geq \mu(k+1)$ on average. So the time it takes will be at most $\int_k^{k+1} \frac{1}{\mu(t)}\,dt$ on average, and the total time is obtained by summing all of these intervals.

Of course, this intuition is not even close to a real proof (among other things, there may be a very dangerous confusion in there between $1/\operatorname{E}[X_n]$ and $\operatorname{E}[1/X_n]$), so we will give a real proof as well.

Proof of Lemma 2.3.1. This is essentially the same proof as in Motwani and Raghavan [MR95], but we add some extra detail to allow for the possibility that $X_n = 0$.

Let $p = \Pr[X_n = 0]$, $q = 1 - p = \Pr[X_n \neq 0]$. Note we have $q > 0$ because otherwise $\operatorname{E}[X_n] = 0 < \mu(n)$. Then we have

$$\begin{aligned}\operatorname{E}[T(n)] &= 1 + \operatorname{E}[T(n - X_n)] \\ &= 1 + p\operatorname{E}[T(n - X_n)|X_n = 0] + q\operatorname{E}[T(n - X_n)|X_n \neq 0] \\ &= 1 + p\operatorname{E}[T(n)] + q\operatorname{E}[T(n - X_n)|X_n \neq 0].\end{aligned}$$

Now we have $\operatorname{E}[T(n)]$ on both sides, which we don't like very much. So we collect it on the left-hand side:

$$(1 - p)\operatorname{E}[T(n)] = 1 + q\operatorname{E}[T(n - X_n)|X_n \neq 0],$$

divide both sides by $q = 1 - p$, and apply the induction hypothesis:

$$\begin{aligned}\operatorname{E}[T(n)] &= 1/q + \operatorname{E}[T(n - X_n)|X_n \neq 0] \\ &= 1/q + \operatorname{E}[\operatorname{E}[T(n - X_n)|X_n]|X_n \neq 0] \\ &\leq 1/q + \operatorname{E}\left[\int_a^{n-X_n} \frac{1}{\mu(t)}\,dt|X_n \neq 0\right] \\ &= 1/q + \operatorname{E}\left[\int_a^{n} \frac{1}{\mu(t)}\,dt - \int_{n-X_n}^{n} \frac{1}{\mu(t)}\,dt|X_n \neq 0\right] \\ &\leq 1/q + \int_a^{n} \frac{1}{\mu(t)}\,dt - \operatorname{E}\left[\frac{X_n}{\mu(n)}|X_n \neq 0\right] \\ &\leq 1/q + \int_a^{n} \frac{1}{\mu(t)}\,dt - \frac{\operatorname{E}[X_n|X_n \neq 0]}{\mu(n)}.\end{aligned}$$

The second-to-last step uses the fact that $\mu(t) \leq \mu(n) for t \leq n$.

It may seem like we don't know what $\operatorname{E}[X_n|X_n \neq 0]$ is. But we know that $X_n \geq 0$, so we have $\operatorname{E}[X_n] = p\operatorname{E}[X_n|X_n = 0] + q\operatorname{E}[X_n|X_n \neq 0] =$

$q \operatorname{E}[X_n | X_n \neq 0]$. So we can solve for $\operatorname{E}[X_n | X_n \neq 0] = E[X_n]/q$. So let's plug this in:

$$\begin{aligned}\operatorname{E}[T(n)] &\leq 1/q + \int_a^n \frac{1}{\mu(t)}\,dt - \frac{\operatorname{E}[X_n]/q}{\mu(n)} \\ &\leq 1/q + \int_a^n \frac{1}{\mu(t)}\,dt - 1/q \\ &= \int_a^n \frac{1}{\mu(t)}\,dt.\end{aligned}$$

This concludes the proof. □

Now we just need to find some applications.

2.3.1 Waiting for heads

For the recurrence $T(1) = 1 + T(1 - X_1)$ with $\operatorname{E}[X_1] = p$, we set $\mu(n) = p$ and get $\operatorname{E}[T(1)] \leq \int_0^1 \frac{1}{p}\,dt = \frac{1}{p}$, which happens to be exactly the right answer.

2.3.2 Quickselect

In Quickselect, we pick a random pivot and split the original array of size n into three piles of size m (less than the pivot), 1 (the pivot itself), and $n-m-1$ (greater than the pivot). We then figure out which of the three piles contains the k-th smallest element (depend on how k compares to $m-1$) and recurse, stopping when we hit a pile with 1 element. It's easiest to analyze this by assuming that we recurse in the largest of the three piles, i.e., that our recurrence is $T(n) = 1 + \max(T(m), T(n-m-1))$, where m is uniform in $0 \ldots n-1$. The exact value of $\operatorname{E}[\max(m, n-m-1)]$ is a little messy to compute (among other things, it depends on whether n is odd or even), but it's not hard to see that it's always less than $(3/4)n$. So letting $\mu(n) = n/4$, we get

$$\operatorname{E}[T(n)] \leq \int_1^n \frac{1}{t/4}\,dt = 4 \ln n.$$

2.3.3 Tossing coins

Here we have $\operatorname{E}[X_n] = (1-p)n$. If we let $\mu(n) = (1-p)n$ and plug into the formula without thinking about it too much, we get

$$\operatorname{E}[T(n)] \leq \int_0^n \frac{1}{(1-p)t}\,dt = \frac{1}{1-p}(\ln n - \ln 0).$$

That $\ln 0$ is trouble. We can fix it by making $\mu(n) = (1-p)\lceil n \rceil$, to get

$$\begin{aligned}\mathrm{E}[T(n)] &\le \int_{0+}^{n} \frac{1}{(1-p)\lceil t \rceil}\, dt \\ &= \frac{1}{1-p}\sum_{k=1}^{n}\frac{1}{k} \\ &= \frac{H_n}{1-p}.\end{aligned}$$

2.3.4 Coupon collector

Now that we know how to avoid dividing by zero, this is easy and fun. Let $\mu(x) = \lceil x \rceil / n$, then we have

$$\begin{aligned}\mathrm{E}[T(n)] &\le \int_{0+}^{n} \frac{n}{\lceil t \rceil}\, dt \\ &= \sum_{k=1}^{n}\frac{n}{k} \\ &= nH_n.\end{aligned}$$

As it happens, this is the exact answer for this case. This will happen whenever X is always a 0–1 variable[1] and we define $\mu(x) = \mathrm{E}[X|n = \lceil x \rceil]$, which can be seen by spending far too much time thinking about the precise sources of error in the inequalities in the proof.

2.3.5 Chutes and ladders

Let $\mu(n)$ be the expected drop from position n. We have to be a little bit careful about small n, but we can compute that in general $\mu(n) = \frac{1}{6}\sum_{i=0}^{\min(n,6)} i$. For fractional values x we will set $\mu(x) = \mu(\lceil x \rceil)$ as before.

[1] A **0–1 random variable** is also called a **Bernoulli random variable**, but 0–1 is shorter to type and more informative. Even more confusing, the underlying experiment that gives rise to a Bernoulli random variable goes by the entirely different name of a **Poisson trial**. Jacob Bernoulli and Siméon-Denis Poisson were great mathematicians, but there are probably better ways to remember them.

Then we have

$$\begin{aligned}\mathrm{E}[T(n)] &\leq \int_{0+}^{n} \frac{1}{\mu(t)}\,dt \\ &= \sum_{k=1}^{n} \frac{1}{\mu(k)}\end{aligned}$$

We can summarize the values in the following table:

n	$\mu(n)$	$1/\mu(n)$	$\sum 1/\mu(k)$
1	1/6	6	6
2	1/2	2	8
3	1	1	9
4	5/3	3/5	48/5
5	5/2	2/5	10
≥ 6	7/2	2/7	$10 + (2/7)(n-5) = (2/7)n + 65/7$

This is a slight overestimate; for example, we can calculate by hand that the expected waiting time for $n = 2$ is 6 and for $n = 3$ that it is $20/3 = 6 + 2/3$.

We can also consider the generalized version of the game where we start at n and drop by $1 \ldots n$ each turn as long as the drop wouldn't take us below 0. Now the expected drop from position k is $k(k+1)/2n$, and so we can apply the formula to get

$$\mathrm{E}[T(n)] \leq \sum_{k=1}^{n} \frac{2n}{k(k+1)}.$$

The sum of $\frac{1}{k(k+1)}$ when k goes from 1 to n happens to have a very nice value; it's exactly $\frac{n}{n+1} = 1 + \frac{1}{n+1}$.[2] So in this case we can rewrite the bound as $2n \cdot \frac{n}{n+1} = \frac{2n^2}{n+1}$.

2.4 High-probability bounds

So far we have only considered bounds on the expected value of $T(n)$. Suppose we want to show that $T(n)$ is in fact small with high probability, i.e.,

[2]Proof: Trivially true for $n = 0$; for larger n, compute $\sum_{k=1}^{n} \frac{1}{k(k+1)} \sum_{k=1}^{n-1} \frac{1}{k(k+1)} + \frac{1}{n(n+1)} = \frac{n-1}{n} + \frac{1}{n(n+1)} = \frac{(n-1)(n+1)-1}{n(n+1)} = \frac{n^2}{n(n+1)} = n/(n+1)$.

a statement of the form $\Pr[T(n) \geq t] \leq \epsilon$. There are two natural ways to do this: we can repeatedly apply Markov's inequality to the expectation bound, or we can attempt to analyze the recurrence in more detail. The first method tends to give weaker bounds but it's easier.

2.4.1 High-probability bounds from expectation bounds

Given $\mathrm{E}[T(n)] \leq m$, we have $\Pr[T(n) \geq \alpha m] \leq \alpha^{-1}$. This does not give a very good bound on probability; if we want to show $\Pr[T(n) \geq t] \leq n^{-c}$ for some constant c (a typical high-probability bound), we need $t \geq n^c m$. But we can get a better bound if m bounds the expected time starting from any reachable state, as is the case for the class of problems we have been considering.

The idea is that if $T(n)$ exceeds αm, we restart the analysis and argue that $\Pr[T(n) \geq 2\alpha m | T(n) \geq \alpha m] \leq \alpha^{-1}$, from which it follows that $\Pr[T(n) \geq 2\alpha m] \leq \alpha^{-2}$. In general, for any non-negative integer k, we have $\Pr[T(n) \geq k\alpha m] \leq \alpha^{-k}$. Now we just need to figure out how to pick α to minimize this quantity for fixed t.

Let $t = k\alpha m$. Then $k = t/\alpha m$ and we seek to minimize $\alpha^{-t/\alpha m}$. Taking the logarithm gives $-(t/m)(\ln \alpha)/\alpha$. The t/m factor is irrelevant to the minimization problem, so we are left with minimizing $-(\ln \alpha)/\alpha$. Taking the derivative gives $-\alpha^{-2} + \alpha^{-2} \ln \alpha$; this is zero when $\ln \alpha = 1$ or $\alpha = e$. (This popular constant shows up often in problems like this.) So we get $\Pr[T(n) \geq kem] \leq e^{-k}$, or, letting $k = \ln(1/\epsilon)$, $\Pr[T(n) \geq em \ln(1/\epsilon)] \leq \epsilon$.

So, for example, we can get an n^{-c} bound on the probability of running too long by setting our time bound to $em \ln(n^c) = cem \ln n = O(m \log n)$. We can't hope to do better than $O(m)$, so this bound is tight up to a log factor.

2.4.2 Detailed analysis of the recurrence

As Lance Fortnow has explained,[3] getting rid of log factors is what theoretical computer science is all about. So we'd like to do better than an $O(m \log n)$ bound if we can. In some cases this is not too hard.

Suppose for each n, $T(n) = 1 + T(X)$, where $\mathrm{E}[X] \leq \alpha n$ for a fixed constant α. Let $X_0 = n$, and let X_1, X_2, etc., be the sequence of sizes of the remaining problem at time 1, 2, etc. Then we have $\mathrm{E}[X_1] \leq \alpha n$ from our assumption. But we also have $\mathrm{E}[X_2] = \mathrm{E}[\mathrm{E}[X_2|X_1]] \leq \mathrm{E}[\alpha X_1] = \alpha \mathrm{E}[X_1] \leq \alpha^2 n$, and by induction we can show that $\mathrm{E}[X_k] \leq \alpha^k n$ for all k. Since

[3] `http://weblog.fortnow.com/2009/01/soda-and-me.html`

X_k is integer-valued, $\mathrm{E}[X_k]$ is an upper bound on $\Pr[X_k > 0]$; we thus get $\Pr[T(n) \geq k] = \Pr[X_k > 0] \leq \alpha^k n$. We can solve for the value of k that makes this less than ϵ: $k = -\log(n/\epsilon)/\log\alpha = \log_{1/\alpha} n + \log_{1/\alpha}(1/\epsilon)$.

For comparison, the bound on the expectation of $T(n)$ from Lemma 2.3.1 is $H(n)/(1-\alpha)$. This is actually pretty close to $\log_{1/\alpha} n$ when α is close to 1, and is not too bad even for smaller α. But the difference is that the dependence on $\log(1/\epsilon)$ is additive with the tighter analysis, so for fixed c we get $\Pr[T(n) \geq t] \leq n^{-c}$ at $t = O(\log n + \log n^c) = O(\log n)$ instead of $O(\log n \log n^c) = O(\log^2 n)$.

2.5 More general recurrences

We didn't do these, but if we had to, it might be worth looking at Roura's Improved Master Theorems [Rou01].

Chapter 3

Probabilistic inequalities

Here we're going to look at some inequalities useful for proving properties of randomized algorithms.

3.1 Basics

Fundamental inequalities for proving more complex inequalities.

3.1.1 Law of total probability

If a set of countably many events $\{A_i\}$ partition the probability space (i.e., they don't intersect each other and their union has probability 1), then for any event B,

$$\Pr[B] = \sum \Pr[B|A_i] \Pr[A_i] \tag{3.1.1}$$

and for any random variable X with finite expectation,

$$\mathrm{E}[X] = \sum \mathrm{E}[X|A_i] \Pr[A_i]. \tag{3.1.2}$$

Here $\Pr[B|A_i] = \frac{\Pr[B \cap A_i]}{\Pr[A_i]}$ is the **conditional probability** of B given A_i, which we can think of as the probability that B occurs if restrict ourself to the subset A_i of the probability space. Similarly, $\mathrm{E}[X|A_i] = \sum_x x \Pr[X = x|A_i]$ is the **conditional expectation** of X given A_i, which again is the expected value of X if we restrict ourselves to A_i.

The law of total probability allows probabilities and expectations to be computed by case analysis, conditioning on each event A_i in turn. Pretty much every analysis of a probabilistic recurrence (see Chapter 2) uses this method.

3.1.2 Union bound (Boole's inequality)

For any countable collection of events $\{A_i\}$,

$$\Pr\left[\bigcup A_i\right] \leq \sum \Pr[A_i]. \tag{3.1.3}$$

Typical use: show that if an algorithm can fail only if various improbable events occur, then the probability of failure is no greater than the sum of the probabilities of these events.

3.1.2.1 Examples

- The proof of Adleman's Theorem in Chapter 5.
- Given a graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, mark a subset of $n/2\sqrt{m}$ vertices uniformly at random. The probability that any particular vertex is marked is then $(n/2/\sqrt{m})/n = 1/2\sqrt{m}$, and the probability that both endpoints of an edge are marked is $(1/2\sqrt{m}) \cdot ((n/2/\sqrt{m} - 1)/n < 1/(2\sqrt{m}))^2 = 1/4m$. So the probability that at least one of the edges has two marked endpoints is at most $m/4m = 1/4$. We thus have a randomized algorithm that outputs an independent set of size $n/2\sqrt{m}$) with probability 3/4, without looking at any of the edges. This is probably not an especially good independent set, but we probably can't hope to do much better without seeing the actual graph.

3.1.3 Linearity of expectation

This says that $\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y]$ and $\mathrm{E}[cX] = c\,\mathrm{E}[X]$ when c is a constant. It also works for countable sums: $\mathrm{E}\left[\sum_i X_i\right] = \sum_i \mathrm{E}[X_i]$ (when the expectations exist and the sum converges). We've already seen this used explicitly in Section 1.2.2 to analyze QuickSort, and it is used implicitly in pretty much all computations of the expected running time of a randomized algorithm, whenever we sum up the expected costs of different parts of the algorithm.

An important feature of linearity of expectation is that it doesn't require independence. For example, a classic problem is to estimate how many runs of heads of length k show up in a sequence of n independent coin-flips. For each starting position $i \in \{0 \ldots n - k\}$, the chance that we get a run of length k starting in that position is exactly 2^{-k}. If we let X_i be the indicator variable for this event, we have $\mathrm{E}[X_i] = 2^{-k}$ and $\mathrm{E}\left[\sum_{i=0}^{n-k} X_i\right] =$

$\sum_{i=0}^{n-k} \mathrm{E}[X_i] = 2^{-k}(n-k+1)$. So, for example, in 1000 coin-flips we expect to see 991/1024 runs of 10 heads on average. This holds even though the X_i are very much not independent: the presence of a run at position X_1 makes it much more likely that there is also a run at position X_2.

On the other hand, this positive correlation means that the high expectation may be the result of rare sequences with many runs instead of common sequences with few runs. So if we care about the probability of seeing at least one length-10 run, we may need different techniques.

3.1.4 Markov's inequality

This is the key tool for turning expectations of non-negative random variables into (upper) bounds on probabilities. Used directly, it generally doesn't give very good bounds, but it can work well if we apply it to $\mathrm{E}[f(X)]$ for a fast-growing function f; see Chebyshev's inequality (3.2.1) or Chernoff bounds (Section 3.2.2).

If $X \geq 0$, then

$$\Pr[X \geq \alpha] \leq \mathrm{E}\, X/\alpha. \tag{3.1.4}$$

The proof is immediate from the Law of Total Probability (3.1.2). We have

$$\begin{aligned}\mathrm{E}[X] &= \mathrm{E}[X|X \geq \alpha] \Pr[X \geq \alpha] + \mathrm{E}[X|x < \alpha] \Pr[X < \alpha] \\ &\geq \alpha \Pr[X \geq \alpha];\end{aligned}$$

now solve for $\Pr[X \geq \alpha]$.

Markov's inequality doesn't work in reverse. For example, consider the following game: for each integer $k > 0$, with probability 2^-k, I give you 2^k dollars. Letting X be your payoff from the game, we have $\Pr[X \geq 2^k] = \sum_{j=k}^{\infty} 2^{-k} = 2^{-k+1} = \frac{2}{2^k}$. The right-hand side here is exactly what we would get from Markov's inequality if $\mathrm{E}[X] = 2$. But in this case, $\mathrm{E}[X] \neq 2$; in fact, the expectation of X is given by $\sum_{k=1} \infty 2^k 2^{-k}$, which diverges.

3.1.4.1 Examples

- The expected running time for randomized QuickSort is $O(n \log n)$. It follows that the probability that randomized QuickSort takes more than $f(n)$ time is $O(n \log n / f(n))$. For example, the probability that it performs the maximum $\binom{n}{2} = O(n^2)$ comparisons is $O(\log n/n)$. (It's possible to do much better than this.)

- Suppose we toss m balls in n bins, uniformly and independently. What is the probability that some particular bin contains at least k balls? The probability that a particular ball lands in a particular bin is $1/n$, so the expected number of balls in the bin is m/n. This gives a bound of m/nk that a bin contains k or more balls. We can combine this with the union bound to show that the probability that any bin contains k or more balls is at most m/k. Unfortunately this is not a very good bound.

- If we flip 1000 coins, we have previously calculated that we expect to see 991/1024 runs of 10 heads in a row (some of which may overlap). So the chance that we get all 1000 coins to come up heads (which would produce exactly 991 runs) is bounded by $\frac{991/1024}{991} = \frac{1}{1024}$. The actual probability is somewhat smaller.

3.1.5 Jensen's inequality

This is mostly useful if we can calculate $\mathrm{E}[X]$ easily for some X, but what we really care about is some other random variable $Y = f(X)$.

If X is a random variable and f is a **convex function**,[1] then

$$f(\mathrm{E}\,X) \leq \mathrm{E}[f(X)]. \tag{3.1.5}$$

See `http://en.wikipedia.org/wiki/Jensen's_inequality` for variants, missing side conditions that exclude pathological f's and X's, and several proofs. The inequality holds in reverse when f is **concave** (i.e., when $-f$ is convex).

3.1.5.1 Examples

- Suppose we flip n fair coins, and we want to get a bound on $\mathrm{E}[X^2]$, where X is the number of heads. The function $f : x \mapsto x^2$ is convex (take its second derivative), so (3.1.5) gives $\mathrm{E}[X^2] \geq (\mathrm{E}\,X)^2 = \frac{n^2}{4}$. (The actual value for $\mathrm{E}[X^2]$ is $n^2/4 + n/4$, which can be found using generating functions.[2])

[1] A function f is **convex** if, for any x, y, and $0 \leq \mu \leq 1$, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$. Geometrically, this means that the line segment between any two points on the graph of f never goes below f; i.e., that the set of points $\{(x, y) | y \geq f(x)\}$ is convex. One way to prove this is to show $\frac{d^2}{dx^2} f \geq 0$.

[2] Here's how: The probability generating function for X is $F(z) = \sum z^k \Pr[X = k] = 2^{-n}(1+z)^n$. Then $zF'(z) = 2^{-n} n z (1+z)^{n-1} = \sum_k k z^k \Pr[X = k]$. Taking the derivative

- For an upper bound, we can choose a concave f. For example, if X is as in the previous example, $\mathrm{E}[\lg X] \leq \lg \mathrm{E}[X] = \lg \frac{n}{2} = \lg n - 1$. This is probably pretty close to the exactly value, as we will see later that X will almost always be within a factor of $1 + o(1)$ of $n/2$. It's not a terribly useful upper bound, because if we use it with (say) Markov's inequality, the most we can prove is that $\Pr[X = n] = \Pr[\lg X = \lg n] \leq \frac{\lg n - 1}{\lg n} = 1 - \frac{1}{\lg n}$, which is an even worse bound than the $1/2$ we can get from applying Markov's inequality to X directly.

3.2 Concentration bounds

If we really want to get tight bounds on a random variable X, the trick will turn out to be picking some non-negative function $f(X)$ where (a) we can calculate $\mathrm{E}[f(X)]$, and (b) f grows fast enough that merely large values of X produce huge values of $f(X)$, allowing us to get small probability bounds by applying Markov's inequality to $f(X)$. This approach is often used to show that X lies close to $\mathrm{E}\,X$ with reasonably high probability, what is known as a **concentration bound**.

3.2.1 Chebyshev's inequality

The simplest concentration bound is **Chebyshev's inequality**, which uses the **variance** of X, given by $\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}\,X)^2] = \mathrm{E}[X^2] - (\mathrm{E}\,X)^2$.

$$\Pr[|X - \mathrm{E}\,X| \geq \alpha] \leq \frac{\mathrm{Var}[X]}{\alpha^2}. \tag{3.2.1}$$

Proof is by applying Markov's inequality to $(X - \mathrm{E}\,X)^2$: for $|X - \mathrm{E}\,X|$ to exceed α, $(X - \mathrm{E}\,X)^2$ must exceed α^2.

This is a weak concentration bound. It's most often useful when X is a sum of random variables, since if $S = \sum_i X_i$, then we can calculate

$$\begin{aligned}\mathrm{Var}[S] &= \sum_{i,j} \mathrm{Cov}(X_i, X_j) \\ &= \sum_i \mathrm{Var}[X_i] + \sum_{i \neq j} \mathrm{Cov}(X_i, X_j),\end{aligned}$$

of this a second time gives $2^{-n}n(1+z)^{n-1} + 2^{-n}n(n-1)z(1+z)^{n-2} = \sum_k k^2 z^{k-1} \Pr[X = k]$. Evaluate this monstrosity at $z = 1$ to get $\mathrm{E}[X^2] = \sum_k k^2 \Pr[X = k] = 2^{-n}n2^{n-1} + 2^{-n}n(n-1)2^{n-2} = n/2 + n(n-1)/4 = \frac{2n+n^2-n}{4} = n^2/4 + n/4$. This is pretty close to the lower bound we got out of Jensen's inequality, but we can't count on this.

where $\mathrm{Var}[X] = \mathrm{E}[X^2] - (\mathrm{E}\,X)^2$ and $\mathrm{Cov}(X,Y) = \mathrm{E}[XY] - \mathrm{E}\,X\,\mathrm{E}\,Y$.

Note that $\mathrm{Cov}(X,Y) = 0$ when X and Y are independent; this makes Chebyshev's inequality particularly useful for **pairwise-independent** random variables. It also works well when the X_i are indicator variables, since if $X \in \{0,1\}$ and $\mathrm{E}[X_i] = p$, we can easily compute $\mathrm{Var}[X_i] = \mathrm{E}[X^2] - (\mathrm{E}\,X)^2 = p - p^2 = p(1-p)$. This is always bounded by p, and for small p the bound is close to tight.

3.2.1.1 Examples

Flipping coins Let X be the sum of n independent fair coins. Let X_i be the indicator variable for the event that the i-th coin comes up heads. Then $\mathrm{Var}[X_i] = 1/4$ and $\mathrm{Var}[X] = \sum \mathrm{Var}[X_i] = n/4$. Chebyshev's inequality gives $\Pr[X = n] \le \Pr[|X - n/2| \ge n/2] \le \frac{n/4}{(n/2)^2} = \frac{1}{n}$. This is still not very good, but it's getting better.

Flipping non-independent coins Let $n = 2^m - 1$ for some m, and let $Y_1 \ldots Y_m$ be independent, fair 0–1 random variables. For each non-empty subset S of $\{1 \ldots m\}$, let X_S be the exclusive OR of all Y_i for $i \in S$. Then (a) the X_i are pairwise independent; (b) each X_i has variance $1/4$; and thus (c) the same Chebyshev's inequality analysis for independent coin flips above applies to $X = \sum_S X_S$, giving $\Pr[|X - n/2| = n/2] \le \frac{1}{n}$. In this case it is not actually possible for X to equal n, but we can have $X = 0$ if all the Y_i are 0, which occurs with probability $2^{-m} = \frac{1}{n+1}$. So the Chebyshev's inequality is about the best we can hope for if we only have pairwise independence.

Balls in bins Let X_i be the indicator that the i-th of m balls lands in a particular bin. Then $\mathrm{E}[X_i] = 1/n$, giving $\mathrm{E}[\sum X_i] = m/n$, and $\mathrm{Var}[X_i] = 1/n - 1/n^2$, giving $\mathrm{Var}[\sum X_i] = m/n - m/n^2$. So the probability that we get $k + m/n$ or more balls in a particular bin is at most $(m/n - m/n^2)/k^2 < m/nk^2$, and applying the union bound, the probability that we get $k + m/n$ or more balls in any of the n bins is less than m/k^2. Setting this equal to ϵ and solving for k gives a probability of at most ϵ of getting more than $m/n + \sqrt{(m/\epsilon)}$ balls in any of the bins. This is still not as good a bound as we can prove, but it's at least non-trivial.

Lazy select This is the example from [MR95, §3.3].

We want to find the k-th smallest element $S_{(k)}$ of a set S of size n. (The parentheses around the index indicate that we are considering the sorted version of the set $S_{(1)} < S_{(2)} \cdots < S_{(n)}$.) The idea is to:

1. Sample a multiset R of $n^{3/4}$ elements of S with replacement and sort them. This takes $O(n^{3/4} \log n^{3/4}) = o(n)$ comparisons so far.

2. Use our sample to find an interval that is likely to contain $S_{(k)}$. The idea is to pick indices $\ell = (k - n^{3/4})n^{-1/4}$ and $r = (k + n^{3/4})n^{-1/4}$ and use $R_{(\ell)}$ and $R_{(r)}$ as endpoints (we are omitting some floors and maxes here to simplify the notation; for a more rigorous presentation see [MR95]). The hope is that the interval $P = [R_{(\ell)}, R(r)]$ in S will both contain $S_{(k)}$, and be small, with $|P| \leq 4n^{3/4} + 2$. We can compute the elements of P in $2n$ comparisons exactly by comparing every element with both $R_{(\ell)}$ and $R_{(r)}$.

3. If both these conditions hold, sort P ($o(n)$ comparisons) and return $S_{(k)}$. If not, try again.

We want to get a bound on how likely it is that P either misses $S_{(k)}$ or is too big.

For any fixed k, the probability that one sample in R is less than or equal to $S_{(k)}$ is exactly k/n, so the expected number X of samples $\leq S_{(k)}$ is exactly $kn^{-1/4}$. The variance on X can be computed by summing the variances of the indicator variables that each sample is $\leq S_{(k)}$, which gives a bound $\mathrm{Var}[X] = n^{3/4}((k/n)(1 - k/n)) \leq n^{3/4}/4$. Applying Chebyshev's inequality gives $\Pr[\left|X - kn^{-1/4}\right| \geq \sqrt{n}] \leq n^{3/4}/4n = n^{-1/4}/4$.

Now let's look at the probability that P misses $S_{(k)}$ because $R_{(\ell)}$ is too big, where $\ell = kn^{-1/4} - \sqrt{n}$. This is

$$\begin{aligned}\Pr[R_{(\ell)} > S_{(k)}] &= \Pr[X < kn^{-1/4} - \sqrt{n}] \\ &\leq n^{-1/4}/4.\end{aligned}$$

(with the caveat that we are being sloppy about round-off errors).

Similarly,

$$\begin{aligned}\Pr[R_{(h)} < S_{(k)}] &= \Pr[X > kn^{-1/4} + \sqrt{n}] \\ &\leq n^{-1/4}/4.\end{aligned}$$

So the total probability that P misses $S_{(k)}$ is at most $n^{-1/4}/2$.

Now we want to show that $|P|$ is small. We will do so by showing that it is likely that $R_{(\ell)} \geq S_{(k-2n^{3/4})}$ and $R_{(h)} \leq S_{(k+2n^{3/4})}$. Let X_ℓ be the number of samples in R that are $\leq S_{(k-2n^{3/4})}$ and X_r be the number of samples in R that are $\leq S_{(k+2n^{3/4})}$. Then we have $\mathrm{E}[X_\ell] = kn^{-1/4} - 2\sqrt{n}$

and $\mathrm{E}[X_r] = kn^{-1/4} + 2\sqrt{n}$, and $\mathrm{Var}[X_l]$ and $\mathrm{Var}[X_r]$ are both bounded by $n^{3/4}/4$.

We can now compute

$$\Pr[R_{(l)} < S_{(k-2n^{3/4})}] = Pr[X_l > kn^{-1/4} - \sqrt{n}] < n^{-1/4}/4$$

by the same Chebyshev's inequality argument as before, and get the symmetric bound on the other side for $\Pr[R_{(r)} > S_{(k+2n^{3/4})}]$. This gives a total bound of $n^{-1/4}/2$ that P is too big, for a bound of $n^{-1/4} = o(n)$ that the algorithm fails on its first attempt.

The total expected number of comparisons is thus given by $T(n) = 2n + o(n) + O(n^{-1/4}T(n)) = 2n + o(n)$.

3.2.2 Chernoff bounds

To get really tight bounds, we apply Markov's inequality to $\exp(\alpha S)$, where $S = \sum_i X_i$. This works best when the X_i are independent: if this is the case, so are the variables $\exp(\alpha X_i)$, and so we get $\mathrm{E}[\exp(\alpha S)] = \mathrm{E}[\prod_i \exp(\alpha X_i)] = \prod_i \mathrm{E}[\exp(\alpha X_i)]$.

The quantity $\mathrm{E}[\exp(\alpha S)]$, treated as a function of α, is called the **moment generating function** of S, because it expands formally into $\sum_k \mathrm{E}[X^k]\alpha^k/k!$, the **exponential generating function** for the series of k-th moments $\mathrm{E}[X^k]$. Note that it may not converge for all S and α;[3] we will be careful to choose α for which it does converge and for which Markov's inequality gives us good bounds.

3.2.2.1 The classic Chernoff bound

The basic Chernoff bound applies to sums of independent 0–1 random variables, which need not be identically distributed. For identically distributed random variables, the sum has a binomial distribution, which we can either compute exactly or bound more tightly using approximations specific to binomial tails; for sums of bounded random variables that aren't necessarily 0–1, we can use Hoeffding's inequality instead (see Section 3.2.3.

Let each X_i for $i = 1 \ldots n$ be a 0–1 random variable with expectation p_i, so that $\mathrm{E}\, S = \mu = \sum_i p_i$. Then $\mathrm{E}[\exp(\alpha X_i)] = p_i e^{\alpha} + (1 - p_i)$, and $\mathrm{E}[\exp(\alpha S)] = \prod_i (p_i e^{\alpha} + 1 - p_i)$. Let $\delta \geq 0$ and $\alpha > 0$. Markov's inequality

[3] For example, the moment generating function for our earlier bad X with $\Pr[X = 2^k] = 2^{-k}$ is equal to $\sum_k 2^{-k} e^{\alpha k}$, which diverges unless $e^{\alpha}/2 < 1$.

then gives

$$\begin{aligned}
\Pr\left[S \geq (1+\delta)\mu\right] &= \Pr\left[e^{\alpha S} \geq \exp\left(\alpha(1+\delta)\mu\right)\right] \\
&\leq \frac{\mathrm{E}[e^{\alpha S}]}{\exp\left(\alpha(1+\delta)\mu\right)} \\
&= \frac{\prod_{i=1}^{n}(p_i e^{\alpha} + 1 - p_i)}{\exp\left(\alpha(1+\delta)\mu\right)} \\
&= \frac{\prod_{i=1}^{n}(1 + p_i(e^{\alpha} - 1))}{\exp\left(\alpha(1+\delta)\mu\right)} \\
&\leq \frac{\prod_{i=1}^{n}\exp\left(p_i(e^{\alpha} - 1)\right)}{\exp\left(\alpha(1+\delta)\mu\right)} \\
&= \frac{\exp\left(\sum_{i=1}^{n} p_i(e^{\alpha} - 1)\right)}{\exp\left(\alpha(1+\delta)\mu\right)} \\
&= \frac{\exp\left(((e^{\alpha} - 1)\mu\right)}{\exp\left(\alpha(1+\delta)\mu\right)} \\
&= \left(\frac{\exp\left(((e^{\alpha} - 1)\right)}{\exp\left(\alpha(1+\delta)\right)}\right)^{\mu}. \\
&= \left(\exp\left(e^{\alpha} - 1 - \alpha(1+\delta)\right)\right)^{\mu}.
\end{aligned}$$

The sneaky inequality step in the middle uses the fact that $(1+x) \leq \exp(x)$ for all x, which itself is one of the most useful inequalities you can memorize.

We now choose α to minimize the base in the last expression, by minimizing $e^{\alpha} - 1 - \alpha(1+\delta)$. Setting the derivative with respect to α to zero gives $e^{\alpha} = (1+\delta)$ or $\alpha = \ln(1+\delta)$. If we plug this back into the bound, we get

$$\begin{aligned}
\Pr\left[S \geq (1+\delta)\mu\right] &\leq \left(\exp\left((1+\delta) - 1 - (1+\delta)\ln(1+\delta)\right)\right)^{\mu} \\
&= \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}. \qquad (3.2.2)
\end{aligned}$$

The base of this rather atrocious quantity is $e^0/1^1 = 1$ at $\delta = 0$, and its derivative is negative for $\delta \geq 0$ (the easiest way to show this is to substitute $\delta = x - 1$ first). So the bound is never greater than 1 and is less than 1 as soon as $\delta > 0$. We also have that the bound is exponential in μ for any fixed δ.

If we look at the shape of the base as a function of δ, we can observe that when δ is very large, we can replace $(1+\delta)^{1+\delta}$ with δ^{δ} without changing the bound much (and to the extent that we change it, it's an increase, so it still

works as a bound). This turns the base into $\frac{e^\delta}{\delta^\delta} = (e/\delta)^\delta = 1/(\delta/e)^\delta$. This is pretty close to Stirling's formula for $1/\delta!$ (there is a $\sqrt{2\pi\delta}$ factor missing).

For very small δ, we have that $1 + \delta \approx e^\delta$, so the base becomes approximately $\frac{e^\delta}{e^{\delta(1+\delta}} = e^{-\delta^2}$. This approximation goes in the wrong direction (it's smaller than the actual value) but with some fudging we can show bounds of the form $e^{-\mu\delta^2/c}$ for various constants c, as long as δ is not too big.

3.2.2.2 Chernoff bound variants

The full Chernoff bound can be difficult to work with, especially since it's hard to invert to find a good δ that gives a particular ϵ bound. Fortunately, there are approximate variants that substitute a weaker but less intimidating bound. Some of the more useful are:

- For $0 \leq \delta \leq 1.81$,

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}. \tag{3.2.3}$$

  (The actual upper limit is slightly higher.) Useful for small values of δ, especially because the bound can be inverted: if we want $\Pr[X \geq (1+\delta)\mu] \leq \exp(-\mu\delta^2/3) \leq \epsilon$, we can use any δ with $\sqrt{3\ln(1/\epsilon)/\mu} \leq \delta \leq 1.81$. The essential idea to the proof is to show that, in the given range, $e^\delta/(1+\delta)^{1+\delta} \leq \exp(-\delta^2/3)$. This is easiest to do numerically; a somewhat more formal argument that the bound holds in the range $0 \leq \delta \leq 1$ can be found in [MU05, Theorem 4.4].

- For $0 \leq \delta \leq 4.11$,

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/4}. \tag{3.2.4}$$

  This is a slightly weaker bound than the previous that holds over a larger range. It gives $\Pr[X \geq (1+\delta)\mu] \leq \epsilon$ if $\sqrt{4\ln(1/\epsilon)/\mu} \leq \delta \leq 4.11$. Note that the version given on page 72 of [MR95] is *not correct*; it claims that the bound holds up to $\delta = 2e - 1 \approx 4.44$, but it fails somewhat short of this value.

- For $R \geq 2e\mu$,

$$\Pr[X \geq R] \leq 2^{-R}. \tag{3.2.5}$$

  Sometimes the assumption is replaced with the stronger $R \geq 6\mu$ (this is the version given in [MU05, Theorem 4.4], for example); one can

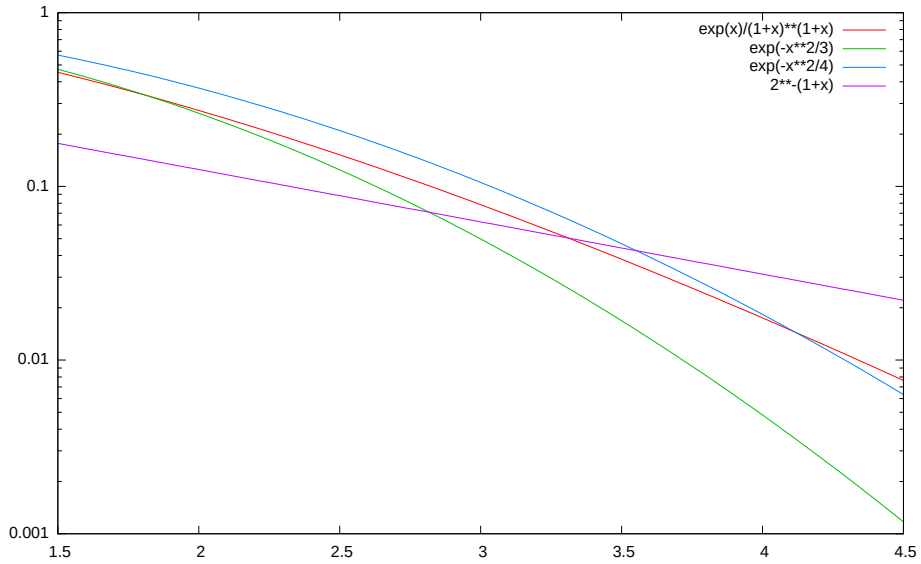


Figure 3.1: Comparison of Chernoff bound variants. The other bounds are valid only in the regions where they exceed $\exp(x)/(1+x)^{1+x}$.

also verify numerically that $R \geq 5\mu$ (i.e., $\delta \geq 4$) is enough. The proof of the $2e\mu$ bound is that $e^\delta/(1+\delta)^{(1+\delta)} < \exp(1+\delta)/(1+\delta)^{(1+\delta)} = (e/(1+\delta))^{1+\delta} \leq 2^{-(1+\delta)}$ when $e/(1+\delta) \leq 1/2$ or $\delta \geq 2e-1$. Raising this to μ gives $\Pr[X \geq (1+\delta)\mu] \leq 2^{-(1+\delta)\mu}$ for $\delta \geq 2e-1$. Now substitute R for $(1+\delta)\mu$ (giving $R \geq 2e\mu$) to get the full result. Inverting this one gives $\Pr[X \geq R] \leq \epsilon$ when $R \geq \min(2e\mu, \lg(1/\epsilon))$.

Figure 3.1 shows the relation between the various bounds when $\mu = 1$, in the region where they cross each other.

3.2.2.3 Lower bound version of Chernoff bounds

We can also use Chernoff bounds to show that a sum of independent 0–1 random variables isn't too small. The essential idea is to repeat the upper bound argument with a negative value of α, which makes $e^{\alpha(1-\delta)\mu}$ an increasing function in δ. The resulting bound is:

$$\Pr[S \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^\mu. \tag{3.2.6}$$

A simpler but weaker version of this bound is

$$\Pr[S \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}. \tag{3.2.7}$$

Both bounds hold for all δ with $0 \leq \delta \leq 1$.

3.2.2.4 Other tail bounds for the binomial distribution

The random graph literature can be a good source for bounds on the binomial distribution. See for example [Bol01, §1.3], which uses normal approximation to get bounds that are slightly tighter than Chernoff bounds in some cases, and [JLR00, Chapter 2], which describes several variants of Chernoff bounds as well as tools for dealing with sums of random variables that aren't fully independent.

3.2.2.5 Applications

Flipping coins Suppose S is the sum of n independent fair coin-flips. Then $\mathrm{E}[S] = n/2$ and $\Pr[S = n] = \Pr[S \geq 2\,\mathrm{E}[S]]$ is bounded using (3.2.2) by setting $\mu = n/2$, $\delta = 1$ to get $\Pr[S = n] \leq (e/4)^{n/2} = (2/\sqrt{e})^{-n}$. This is not quite as good as the real answer 2^{-n} (the quantity $2/\sqrt{e}$ is about $1.213\ldots$), but it's at least exponentially small.

Balls in bins again Let's try applying the Chernoff bound to the balls-in-bins problem. Here we let $S = \sum_{i=1}^{m} X_i$ be the number of balls in a particular bin, with X_i the indicator that the i-th ball lands in the bin, $\mathrm{E}\,X_i = p_i = 1/n$, and $\mathrm{E}\,S = \mu = m/n$. To get a bound on $\Pr[S \geq m/n + k]$, apply the Chernoff bound with $\delta = kn/m$ to get

$$\begin{aligned}\Pr[S \geq m/n + k] &= \Pr[S \geq (m/n)(1 + kn/m)] \\ &\leq \frac{e^k}{(1+kn/m)^{1+kn/m}}.\end{aligned}$$

For $m = n$, this collapses to the somewhat nicer but still pretty horrifying $e^k/(k+1)^{k+1}$.

Staying with $m = n$, if we are bounding the probability of having large bins, we can use the 2^{-R} variant to show that the probability that any particular bin has more than $2\lg n$ balls (for example), is at most n^{-2}, giving the probability that there exists such a bin of at most $1/n$. This is not as strong as what we can get out of the full Chernoff bound. If we

take the logarithm of $e^k/(k+1)^{k+1}$, we get $k-(k+1)\ln(k+1)$; if we then substitute $k = \frac{c\ln n}{\ln\ln n} - 1$, we get

$$\begin{aligned}
&\frac{c\ln n}{\ln\ln n} - 1 - \frac{c\ln n}{\ln\ln n}\ln\frac{c\ln n}{\ln\ln n} \\
&= (\ln n)\left(\frac{c}{\ln\ln n} - \frac{1}{\ln n} - \frac{c}{\ln\ln n}(\ln c + \ln\ln n - \ln\ln\ln n)\right) \\
&= (\ln n)\left(\frac{c}{\ln\ln n} - \frac{1}{\ln n} - \frac{c\ln c}{\ln\ln n} - c + \frac{c\ln\ln\ln n}{\ln\ln n}\right) \\
&= (\ln n)(-c + o(1)).
\end{aligned}$$

So the probability of getting more than $c\ln n/\ln\ln n$ balls in any one bin is bounded by $\exp((\ln n)(-c+o(1))) = n^{-c+o(1)}$. This gives a maximum bin size of $O(\log n/\log\log n)$ with any fixed probability bound n^{-a} for sufficiently large n.

Flipping coins, central behavior Suppose we flip n fair coins, and let S be the number that come up heads. We expect $\mu = n/2$ heads on average. How many extra heads can we get, if we want to stay within a probability bound of n^{-c}?

Here we use the small-δ approximation, which gives $\Pr[S \geq (1+\delta)(n/2)] \leq \exp(-\delta^2 n/6)$. Setting $\exp(-\delta^2 n/6) = n^{-c}$ gives $\delta = \sqrt{6\ln n^c/n} = \sqrt{6c\ln n/n}$. The actual excess over the mean is $\delta(n/2) = (n/2)\sqrt{6c\ln n/n} = \sqrt{\frac{3}{2}cn\ln n}$. By symmetry, the same bound applies to extra tails. So if we flip 1000 coins and see more than 676 heads (roughly the bound when c=3), we can reasonably conclude that either (a) our coin is biased, or (b) we just hit a rare one-in-a-billion jackpot.

In algorithm analysis, the $\sqrt{(3/2)c}$ part usually gets absorbed into the asymptotic notation, and we just say that with probability at least $1-n^{-c}$, the sum of n random bits is within $O(\sqrt{n\log n})$ of $n/2$.

Valiant's randomized hypercube routing Here we use Chernoff bounds to show bounds on a classic **permutation-routing** algorithm for **hypercube networks** due to Valiant [Val82]. The presentation is based on §§4.2 of [MR95], which in turn is based on an improved version of Valiant's original analysis appearing in a follow-up paper with Brebner [VB81].

The basic idea of a hypercube architecture is that we have a collection of $N = 2^n$ processors, each with an n-bit address. Two nodes are adjacent if their addresses differ by one bit. (These things were the cat's pajamas back in the 1980s; see `http://en.wikipedia.org/wiki/Connection_Machine`.)

Suppose that at some point in a computation, each processor i wants to send a packet of data to some processor $\pi(i)$, where π is a permutation of the addresses. But we can only send one packet per time unit along each of the n edges leaving a processor.[4] How do we route the packets so that all of them arrive in the minimum amount of time?

We could try to be smart about this, or we could use randomization. Valiant's idea is to first route each packet to some uniform, independent random intermediate node $\sigma(i)$, then route each packet from $\sigma(i)$ to its ultimate destination $\pi(i)$. Routing is done by a **bit-fixing**: if a packet is currently at node x and heading for node y, find the leftmost bit j where $x_j \neq y_j$ and fix it, by sending the packet on to $x[x_j/y_j]$. In the absence of contention, bit-fixing routes a packet to its destination in at most n steps. The hope is that the randomization will tend to spread the packets evenly across the network, reducing the contention for edges enough that actual time will not be much more than this.

The first step is to argue that, on its way to its random destination, any particular packet is delayed at most one time unit by any other packet whose path overlaps with it. Suppose packet i is delayed by contention on some edge uv. Then there must be some other packet j that crosses uv in the current round. From this point on, j remains one step ahead of i (until its path diverges), so it can't block i again unless both are blocked by some third packet k (in which case we charge i's further delay to k). This means that we can bound the delays for packet i by counting how many other packets cross its path.[5] So now we just need a high-probability bound on the number of packets that get in my way.

Following the presentation in [MR95], define H_{ij} to be the indicator variable for the event that packets i and j cross paths. To keep things symmetric, we'll throw in $H_{ii} = 1$ as well (this only slightly increases the bound). Because each j chooses its destination independently, once we fix i's path, the H_{ij} are all independent. So we can bound $S = \sum_j H_{ij}$ using Chernoff bounds. To do so, we must first calculate an upper bound on $\mu = \mathrm{E}\, S$.

The trick here is to observe that any path that crosses i's path must cross one of its edges. Let T_e be the number of paths that cross edge e, and X_i be the number of edges that path i crosses. Counting two ways, we have $\sum_e T_e = \sum_i X_i$, and so $\mathrm{E}[\sum_e T_e] = \mathrm{E}[\sum_i X_i] = N(n/2)$. By symmetry,

[4] Formally, we have a synchronous routing model with unbounded buffers at each node, with a maximum capacity of one packet per edge per round.

[5] A much more formal version of this argument is given as [MR95, Lemma 4.5].

all the T_e have the same expectation, so we get $\mathrm{E}[T_e] = \frac{N(n/2)}{Nn} = 1/2$. But now we can sum $\mathrm{E}[T_e]$ for the at most n edges e on i's path, to get $\mathrm{E}[\sum_j H_{ij}] \leq n/2$—on average, i collides with at most $n/2$ other packets.

Inequality (3.2.5) says that $\Pr[X \geq R] \leq 2^{-R}$ when $R \geq 2e\mu$. Setting $R = 3n$ gives $R \geq 6\mu \geq 2e\mu$, so $\Pr[\sum_j H_{ij} \geq 3n] \leq 2^{-3n} = N^{-3}$. This says that any one packet reaches its random destination with at most $3n$ added delay (thus, in at most $4n$ time units) with probability at least $1 - N^{-3}$. If we consider all N packets, the total probability that any of them fail to reach their random destinations in $4n$ time units is at most $N \cdot N^{-3} = N^{-2}$.

What about the second phase? Here, routing the packets from the random destinations back to the real destinations is just the reverse of routing them from the real destinations to the random destinations. So the same bound applies, and with probability at most N^{-2} some packet takes more than $4n$ time units to get back (this assumes that we hold all the packets before sending them back out, so there are no collisions between packets from different phases).

Adding up the failure probabilities and costs for both stages gives a probability of at most $2/N^2$ that any packet takes more than $8n$ time units to reach its destination.

The structure of this argument is pretty typical for applications of Chernoff bounds: we get an exponentially small bound on the probability that something bad happens by applying Chernoff bounds to a part of the problem where we have independence, then use the union bound to extend this to the full problem where we don't.

3.2.3 Azuma-Hoeffding inequality

Sometimes we want to get an exponential probability bound for a sum of small random variables (as when using Chernoff bounds) but we don't have full independence. In the worst case we are out of luck (suppose all of our random variables are equal), but in many cases we can use the **Azuma-Hoeffding inequality**, which is often called **Azuma's inequality** when applied to bounded-difference martingales or **Hoeffding's inequality** when applied to sums of bounded independent random variables.[6]

[6]The history of this is that Hoeffding [Hoe63] proved it for independent random variables, and observed that the proof was easily extended to martingales, while Azuma [Azu67] actually went and did the work of proving it for martingales. It's really the same bound, just facing in different directions.

3.2.3.1 Hoeffding's inequality

We'll start by proving the version for independent random variables, which is a little easier. We can then show that it continues to work for martingales to get the full inequality.

Theorem 3.2.1. *Let $X_1 \dots X_n$ be independent random variables with $\mathrm{E}[X_i] = 0$ and $|X_i| \leq c_i$ for all i. Then for all t,*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq t\right] \leq \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{3.2.8}$$

Proof. Let $S = \sum_{i=1}^{n} X_i$. As with Chernoff bounds, we'll first calculate a bound on the moment generating function $\mathrm{E}\left[e^{\alpha S}\right]$ and then apply Markov's inequality with a carefully-chosen α.

Before jumping into this, it helps to get a bound on $\mathrm{E}\left[e^{\alpha X}\right]$ when $\mathrm{E}[X] = 0$ and $|X| \leq c$. The trick is that, for any α, $e^{\alpha x}$ is a convex function. Since we want an upper bound, we can't use Jensen's inequality (3.1.5), but we *can* use the fact that X is bounded and we know its expectation. Convexity of $e^{\alpha x}$ means that, for any x with $-c \leq x \leq c$, $e^{\alpha x} \leq \lambda e^{-\alpha c} + (1-\lambda)e^{\alpha c}$, where $x = \lambda(-c) + (1-\lambda)c$. Solving for λ in terms of x gives $\lambda = \frac{1}{2}\left(1 - \frac{x}{c}\right)$ and $1 - \lambda = \frac{1}{2}\left(1 + \frac{x}{c}\right)$. So

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha X}\right] &\leq \mathrm{E}\left[\frac{1}{2}\left(1 - \frac{X}{c}\right)e^{-\alpha c} + \frac{1}{2}\left(1 + \frac{X}{c}\right)e^{\alpha c}\right] \\
&= \frac{e^{-\alpha c} + e^{\alpha c}}{2} - \frac{e^{-\alpha c}}{2c}\mathrm{E}\,X + \frac{e^{\alpha c}}{2c}\mathrm{E}\,X \\
&= \frac{e^{-\alpha c} + e^{\alpha c}}{2} \\
&= \cosh(\alpha c).
\end{aligned}$$

In other words, the worst possible X is a fair choice between $\pm c$, and in this case we get the hyperbolic cosine of αc as its moment generating function.

We don't like hyperbolic cosines much, because we are going to want to take products of our bounds, and hyperbolic cosines don't multiply very nicely. As before with $1 + x$, we'd be much happier if we could replace the cosh with a nice exponential. The Taylor's series expansion of $\cosh x$ starts with $1 + x^2/2 + \dots$, suggesting we approximate it with $\exp(x^2/2)$, and indeed it is the case that for all x, $\cosh x \leq e^{x^2/2}$. This can be shown by comparing

the rest of the Taylor's series expansions:

$$\begin{aligned}
\cosh x &= \frac{e^x + e^{-x}}{2} \\
&= \frac{1}{2}\left(\sum_{n=0}^{\infty} \frac{x^n}{n!} + \sum_{n=0}^{\infty} \frac{(-x)^n}{n!}\right) \\
&= \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} \\
&\leq \sum_{n=0}^{\infty} \frac{x^{2n}}{2^n n!} \\
&= \sum_{n=0}^{\infty} \frac{(x^2/2)^n}{n!} \\
&= e^{x^2/2}.
\end{aligned}$$

This gives $\mathrm{E}[e^{\alpha X}] \leq e^{(\alpha c)^2/2}$.

Using this bound and the independence of the X_i, we compute

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha S}\right] &= \mathrm{E}\left[\exp\left(\alpha \sum_{i=1}^{n} X_i\right)\right] \\
&= \mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right] \\
&= \prod_{i=1}^{n} \mathrm{E}\left[e^{\alpha X_i}\right]. \\
&\leq \prod_{i=1}^{n} e^{(\alpha c_i)^2/2} \\
&= \exp\left(\sum_{i=1}^{n} \frac{\alpha^2 c_i^2}{2}\right) \\
&= \exp\left(\frac{\alpha^2}{2} \sum_{i=1}^{n} c_i^2\right).
\end{aligned}$$

Applying Markov's inequality then gives (when $\alpha > 0$):

$$\begin{aligned}
\Pr\left[S \geq t\right] &= \Pr\left[e^{\alpha S} \geq e^{\alpha t}\right] \\
&\leq \exp\left(\frac{\alpha^2}{2} \sum_{i=1}^{n} c_i^2 - \alpha t\right). \qquad (3.2.9)
\end{aligned}$$

Now we do the same trick as in Chernoff bounds and choose α to minimize the bound. If we write C for $\sum_{i=1}^{n} c_i^2$, this is done by minimizing the exponent $\frac{\alpha^2}{2}C - \alpha t$, which we do by taking the derivative with respect to α and setting it to zero: $\alpha C - t = 0$, or $\alpha = t/C$. At this point, the exponent becomes $\frac{(t/C)^2}{2}C - (t/C)t = -\frac{t^2}{2C}$.

Plugging this into (3.2.9) gives the bound (3.2.8) claimed in the theorem. □

Let's see how good a bound this gets us for our usual test problem of bounding $\Pr[S = n]$ where $S = \sum_{i=1}^{n} X_i$ is the sum of n independent fair coin-flips. To make the problem fit the theorem, we replace each X_i by a rescaled version $Y_i = 2X_i - 1 = \pm 1$ with equal probability; this make $\mathrm{E}[Y_i] = 0$ as needed, with $|Y_i| \le c_i = 1$. Hoeffding's inequality (3.2.8) then gives

$$\begin{aligned}\Pr\left[\sum_{i=1}^{n} Y_i \ge n\right] &\le \exp\left(-\frac{n^2}{2n}\right) \\ &= e^{-n/2} = (\sqrt{e})^{-n}.\end{aligned}$$

Since $\sqrt{e} \approx 1.649\ldots$, this is actually slightly better than the $(2/\sqrt{e})^{-n}$ bound we get using Chernoff bounds.

On the other hand, Chernoff bounds work better if we have a more skewed distribution on the X_i; for example, in the balls-in-bins case, each X_i is a 0–1 random variable with $\mathrm{E}[X_i] = 1/n$. Using Hoeffding's inequality, we get a bound c_i on $|X_i - \mathrm{E}[X_i]|$ of only $1 - 1/n$, which puts $\sum_{i=1}^{n} c_i^2$ very close to n, requiring $t = \Omega(\sqrt{n})$ before we get any non-trivial bound out of (3.2.8), pretty much the same as in the fair-coin case (which is not surprising, since Hoeffding's inequality doesn't know anything about the distribution of the X_i). But we've already seen that Chernoff gives us that $\sum X_i = O(\log n / \log\log n)$ with high probability in this case.

Note: There is an asymmetrical version of Hoeffding's inequality in which $a_i \le X_i \le b_i$, but $\mathrm{E}[X_i]$ is still zero for all X_i. In this version, the bound is

$$\Pr\left[\sum_{i=1}^{n} X_i \ge t\right] \le \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{3.2.10}$$

This reduces to (3.2.8) when $a_i = -c_i$ and $b_i = c_i$. The proof is essentially the same, but a little more analytic sneakery is required to show that $\mathrm{E}[e^{\alpha X_i}] \le e^{\alpha^2 (b_i - a_i)^2/8}$; see [McD89] for details. For most applications, the only difference between the symmetric version (3.2.8) and the asymmetric version (3.2.10) is a small constant factor on the resulting bound on t.

3.2.3.2 Azuma's inequality

A general rule of thumb is that most things that work for sums of independent random variables also work for martingales.

A **martingale** is a sequence of random variables $S_1, S_2, \ldots$, where $\mathrm{E}[S_t|S_1, \ldots, S_{t-1}] = S_{t-1}$. In other words, given everything you know up until time $t-1$, your best guess of the expected value at time t is just wherever you are now.

Another way to describe a martingale is to take the partial sums $S_t = \sum_{i=1}^{t} X_t$ of a **martingale difference sequence**, which is a sequence of random variables $X_1, X_2, \ldots$ where $E[X_t|X_1 \ldots X_{t-1}] = 0$. So in this version, your expected change from time $t-1$ to t averages out to zero, even if you try to predict it using all the information you have at time $t-1$.

Martingales were invented to analyze fair gambling games, where your return over some time interval is not independent of previous outcomes (for example, you may change your bet or what game you are playing depending on how things have been going for you), but it is always zero on average given previous information (warning: real casinos give negative expected return, so the resulting process is a **supermartingale** with $S_{t-1} \geq \mathrm{E}[S_t|S_0 \ldots S_{t-1}]$). The nice thing about martingales is they allow for a bit of dependence while still acting very much like sums of independent random variables.

Where this comes up with Hoeffding's inequality is that we might have a process that is reasonably well-behaved, but its increments are not technically independent. For example, suppose that a gambler plays a game where she bets x units $0 \leq x \leq 1$ at each round, and receives $\pm x$ with equal probability. Suppose also that her bet at each round may depend on the outcome of previous rounds (for example, she might stop betting entirely if she loses too much money). If X_i is her take at round i, we have that $\mathrm{E}[X_i|X_1 \ldots X_{i-1}] = 0$ and that $|X_i| \leq 1$. This is enough to apply the martingale version of Hoeffding's inequality, often called Azuma's inequality.

Theorem 3.2.2. *Let $\{S_k\}$ be a martingale with $S_k = \sum_{i=1}^{t} X_i$ and $|X_i| \leq c_i$ for all i. Then for all n and all $t \geq 0$:*

$$\Pr\left[S_n \geq t\right] \leq \exp\left(\frac{-t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{3.2.11}$$

Proof. Basically, we just show that $\mathrm{E}\left[e^{\alpha S_n}\right] \leq \exp\left(\frac{\alpha^2}{2}\sum_{i=1}^{n} c_i^2\right)$—just like in the proof of Theorem 3.2.1—and the rest follows using the same argument. The only tricky part is we can no longer use independence to transform $\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right]$ into $\prod_{i=1}^{n} \mathrm{E}\left[e^{\alpha X_i}\right]$.

Instead, we use the martingale property. For each X_i, we have $\mathrm{E}[X_i|X_1 \ldots X_{i-1}] = 0$ and $|X_i| \leq c_i$ always. Recall that $\mathrm{E}[e^{\alpha X_i}|X_1 \ldots X_{i-1}]$ is a random variable that takes on the average value of $e^{\alpha X_i}$ for each setting of $X_1 \ldots X_{i-1}$; we can apply the same analysis as in the proof of 3.2.1 to show that this means that $\mathrm{E}[e^{\alpha X_i}|X_1 \ldots X_{i-1}] \leq e^{(\alpha c_i)^2/2}$ always.

The trick is to use the fact that, for any random variables X and Y, $\mathrm{E}[XY] = \mathrm{E}[\mathrm{E}[XY|X]] = \mathrm{E}[X\,\mathrm{E}[Y|X]]$.

We argue by induction on n that $\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right] \leq \prod_{i=1}^{n} e^{(\alpha c)^2/2}$. The base case is when $n = 0$. For the induction step, compute

$$\begin{aligned}
\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right] &= \mathrm{E}\left[\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}|X_1 \ldots X_{n-1}\right]\right] \\
&= \mathrm{E}\left[\left(\prod_{i=1}^{n} e^{\alpha X_i}\right)\mathrm{E}\left[e^{\alpha X_n}|X_1 \ldots X_{n-1}\right]\right] \\
&\leq \mathrm{E}\left[\left(\prod_{i=1}^{n-1} e^{\alpha X_i}\right)e^{(\alpha c_n)^2/2}\right] \\
&= \mathrm{E}\left[\prod_{i=1}^{n-1} e^{\alpha X_i}\right]e^{(\alpha c_n)^2/2} \\
&\leq \left(\prod_{i=1}^{n-1} e^{(\alpha c_i)^2/2}\right)e^{(\alpha c_n)^2/2} \\
&= \prod_{i=1}^{n} e^{(\alpha c_i)^2/2}. \\
&= \exp\left(\frac{\alpha^2}{2}\sum_{i=1}^{n} c_i^2\right).
\end{aligned}$$

The rest of the proof goes through as before. □

Some extensions:

- The same bound works for bounded-difference supermartingales. If $\mathrm{E}[X_i|X_1 \ldots X_{i-1}] \leq 0$ and $|X_i| \leq c_i$, then we can write $X_i = Y_i + Z_i$ where $Y_i = \mathrm{E}[X_i|X_1 \ldots X_{i-1}] \leq 0$ is predictable from $X_1 \ldots X_{i-1}$ and $\mathrm{E}[Z_i|X_1 \ldots X_{i-1}] = 0$. Then we can bound $\sum_{i=1}^{n} X_i$ by observing that it is no greater than $\sum_{i=1}^{n} Z_i$. A complication is that we no longer have $|Z_i| \leq c_i$; instead, $|Z_i| \leq 2c_i$ (since leaving out Y_i may shift Z_i

up). But with this revised bound, (3.2.11) gives

$$\Pr\left[\sum_{i=1}^{n} X_i \geq t\right] \leq \Pr\left[\sum_{i=1}^{n} Z_i \geq t\right] \leq \exp\left(-\frac{t^2}{8\sum_{i=1}^{n} c_i^2}\right). \tag{3.2.12}$$

- Suppose that we stop the process after the first time τ with $S_\tau = \sum_{i=1}^{\tau} X_i \geq t$. This is equivalent to making a new variable Y_i that is zero whenever $S_{i-1} \geq t$ and equal to X_i otherwise. This doesn't affect the conditions $\mathrm{E}[Y_i|Y_1 \ldots Y_{i-1}] = 0$ or $|Y_i| \leq c_i$, but it makes it so $\sum_{i=1}^{n} Y_i \geq t$ if and only if $\max_{k\leq n} \sum_{i=1}^{k} X_i \geq t$. Applying (3.2.11) to $\sum Y_i$ then gives

$$\Pr\left[\max_{k\leq n} \sum_{i=1}^{k} X_i \geq t\right] \leq \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{3.2.13}$$

- Since the conditions on X_i in Theorem 3.2.2 apply equally well to $-X_i$, we have

$$\Pr\left[\sum_{i=1}^{n} X_i \leq -t\right] \leq \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{3.2.14}$$

  which we can combine with (3.2.11) to get the two-sided bound

$$\Pr\left[\left|\sum_{i=1}^{n} X_i\right| \geq t\right] \leq 2\exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{3.2.15}$$

- The extension of Hoeffding's inequality to the case $a_i \leq X_i \leq b_i$ works equally well for Azuma's inequality, giving the same bound as in (3.2.10).

- Finally, one can replace the requirement that each c_i be a constant with a requirement that c_i be predictable from $X_1 \ldots X_{i-1}$ and that $\sum_{i=1}^{n} c_i^2 \leq C$ always and get $\Pr\left[\sum_{i=1}^{n} X_i \geq t\right] \leq e^{-t^2/2C}$. This geneally doesn't come up unless you have an algortihm that explicitly cuts off the process if $\sum c_i^2$ gets too big, but there is at least one example of this in the literature [AW96].

3.2.3.3 The method of bounded differences

To use Azuma's inequality, we need a bounded-difference martingale. The easiest way to get such martingales is through the **method of bounded differences**, which was popularized by a survey paper by McDiarmid [McD89]. For this reason the key result is often referred to as **McDiarmid's inequality**.

The basic idea of the method is to imagine we are computing a function $f(X_1, \ldots, X_n)$ of a sequence of independent random variables $X_1, \ldots, X_n$. To get our martingale, we'll imagine we reveal the X_i one at a time, and compute at each step the expectation of the final value of f based on just the inputs we've seen so far.

Formally, let $Y_t = \mathrm{E}[f|X_1, \ldots, X_t]$, the expected value of f given the values of the first t variables. Then $\{Y_t\}$ forms a martingale, with $Y_0 = \mathrm{E}[f]$ and $Y_n = \mathrm{E}[f|X_1, \ldots, X_t] = f$.[7] So if we can find a bound c_t on $Y_t - Y_{t-1}$, we can apply Azuma's inequality to get bounds on $Y_n - Y_0 = f - \mathrm{E}[f]$.

We do this by assuming that f is **Lipschitz** with respect to **Hamming distance** on its domain.[8] This means that there are bounds c_t such that for any $x_1 \ldots x_n$ and any x'_t, we have

$$\left|f(x_1 \ldots x_t \ldots x_n) - f(x_1 \ldots x'_t \ldots x_n)\right| \leq c_t. \tag{3.2.16}$$

Lemma 3.2.3. *If f is Lipschitz with bounds c_i, then $|\mathrm{E}[f|X_1, \ldots, X_t] - \mathrm{E}[f|X_1, \ldots, X_{t-1}]| \leq c_t$.*

[7] A sequence of random variables of the form $Y_t = \mathrm{E}[Z|X_1, \ldots, X_t]$ is called a **Doob martingale**. The proof that it is a martingale is slightly painful using the tools we've got, but the basic idea is that we can expand $\mathrm{E}[Y_t|X_1, \ldots, X_{t-1}] = \mathrm{E}[\mathrm{E}[Z|X_1, \ldots, X_t]|X_1, \ldots, X_{t-1}] = \mathrm{E}[Z|X_1, \ldots, X_{t-1}] = Y_{t-1}$, where the step in the middle follows from the usual repeated-averaging trick that shows $\mathrm{E}[\mathrm{E}[X|Y, Z]|Z] = \mathrm{E}[X|Z]$. To change the variables we are conditioning Y_t on from X_i's to Y_i's, we have to observe that the X_i give at least as much information as the corresponding Y_i (since we can calculate $Y_1, \ldots, Y_{t-1}$ from $X_1, \ldots, X_{t-1}$), so $\mathrm{E}[Y_t|Y_1, \ldots, Y_{t-1}] = \mathrm{E}[\mathrm{E}[Y_t|X_1, \ldots, X_{t-1}]|Y_1, \ldots, Y_{t-1}] = \mathrm{E}[Y_{t-1}|Y_1, \ldots, Y_{t-1}] = Y_{t-1}$. This establishes the martingale property.

[8] The Hamming distance between two vectors is the number of places where they differ, without regard to how much they differ by in these places. A Lipschitz function in general is a function f such that there exists a constant c for which $d(f(x), f(y)) \leq cd(x, y)$ for all x, y in f's domain, where d is the distance between two points in our preferred metrics for the domain and codomain. This is similar to but stronger than the usual notion of continuity.

Proof. Fix $x_1, \ldots, x_t$, and suppose $X_i = x_i$ for $1 \le i \le t$. Then

$$\begin{aligned}
&\mathrm{E}[f|X_1, \ldots, X_{t-1}] - \mathrm{E}[f|X_1, \ldots, X_{t-1}] \\
&= \sum_{x'_t, x_{t+1}, \ldots x_n} (f(x_1, \ldots, x_t, x_{t+1}, \ldots, x_n) - f(x_1, \ldots, x'_t, x_{t+1}, \ldots, x_n)) \Pr[X_t = x'_t] \prod_{j=t+1}^{n} \Pr[X_j = x_j] \\
&\le \sum_{x'_t, x_{t+1}, \ldots x_n} c_i \Pr[X_t = x'_t] \prod_{j=t+1}^{n} \Pr[X_j = x_j] \\
&= c_i.
\end{aligned}$$

A symmetric argument shows a lower bound of $-c_i$, giving the full bound. □

Theorem 3.2.4. *Let $X_1, \ldots, X_n$ be independent random variables and let $f(X_1, \ldots, X_n)$ be Lipschitz with bounds c_i. Then*

$$\Pr[f(X_1, \ldots, X_n) - \mathrm{E}[f(X_1, \ldots, X_n)] \ge t] \le \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \quad (3.2.17)$$

Proof. Lemma 3.2.3 establishes that the martingale $Y_t = \mathrm{E}[f|X_1, \ldots, X_t]$ has bounded differences with bound sequence c_i. Azuma's inequality (3.2.11) immediately gives the stated bound. □

Since we applying Azuma's inequality to a martingale, the lower bound from (3.2.14) and the two-sided bound from (3.2.15) also apply.

3.2.3.4 Applications

Here are some applications of the preceding inequalities. Most of these are examples of the method of bounded differences.

- Suppose you live in a hypercube, and the local government has conveniently placed snow shovels on some subset A of the nodes. If you start at a random location, how likely is it that your distance to the nearest snow shovel deviates substantially from the average distance?

  We can describe your position as a bit vector $X_1, \ldots, X_n$, where each X_i is an independent random bit. Let $f(X_1, \ldots, X_n)$ be the distance from $X_1, \ldots, X_n$ to the nearest element of A. Then changing one of the bits changes the distance by at most 1. So we have $\Pr[|f - \mathrm{E}[f]| \ge t] \le 2e^{-t^2/2n}$ by (3.2.17), giving a range of distances that is $O(\sqrt{n \log n})$

with probability at least $1 - n^{-c}$. Of course, without knowing what A is, we don't know what $E[f]$ is; but at least we can be assured that (unless A is very big) the distance we have to walk through the snow will be pretty much the same pretty much wherever we start.

- Consider a **random graph** $G(n, p)$ consisting of n vertices, where each possible edge appears with independent probability p. Let χ be the **chromatic number** of this graph, the minimum number of colors necessary if we want to assign a color to each vertex that is distinct for the colors of all of its neighbors. The **vertex exposure martingale** shows us the vertices of the graph one at a time, along with all the edges between vertices that have been exposed so far. We define X_t to be the expected value of χ given this information for vertices $1 \ldots t$.

  If Z_i is a random variable describing which edges are present between i and vertices less than i, then the Z_i are all independent, and we can write $\chi = f(Z_1, \ldots, Z_n)$ for some function f (this function may not be very easy to computed, but it exists). Then X_t as defined above is just $\mathrm{E}[f|Z_1, \ldots, Z_t]$. Now observe that f is Lipschitz with $c_t = 1$: if I change the edges for some vertex v_t, I can't increase the number of colors I need by more than 1, since in the worst case I can always take whatever coloring I previously had for all the other vertices and add a new color for v_t. (This implies I can't decrease χ by more than one either.) McDiarmid's inequality (3.2.17) then says that $\Pr[|\chi - \mathrm{E}[\chi]| \geq t] \leq 2e^{-t^2/2n}$; in other words, the chromatic number of a random graph is tightly concentrated around its mean, even if we don't know what that mean is. (This proof is due to Shamir and Spencer [SS87].)

- Suppose we toss m balls into n bins. How many empty bins do we get? The probability that each bin individually is empty is exactly $(1 - 1/n)^m$, which is approximately $e^{-m/n}$ when n is large. So the expected number of empty bins is exactly $n(1-1/n)^m$. If we let X_i be the bin that ball i gets tossed into, and let $Y = f(X_1, \ldots, X_m)$ be the number of empty bins, then changing a single X_i can change f by at most 1. So from (3.2.17) we have $\Pr[Y \geq n(1 - 1/n)^m + t] \leq e^{-t^2/2m}$.

- Most probabilistic recurrence arguments (as in Chapter 2) can be interpreted as supermartingales (my current estimate of $T(n)$ exceeds the expected estimate after I do one stage of the recurrence). This fact can be used to get concentration bounds using (3.2.12).

For example, let's take the recurrence (1.2.1) for the expected number of comparisons for QuickSort:

$$T(n) = \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-1-k)\right).$$

We showed in Section 1.2.1 that the solution to this recurrence satisfies $T(n) \le 2n \ln n$.

To turn this into a supermartingale, imagine that we carry out a process where we keep around at each step t a set of unsorted blocks of size $n_1^t, n_2^t, \ldots, n_{k_t}^t$ for some k_t (note that the superscripts on n_i^t are not exponents). One step of the process involves choosing one of the blocks (we can do this arbitrarily without affecting the argument) and then splitting that block around a uniformly-chosen pivot. We will track a random variable X_t equal to $C_t + \sum_{i=1} k_t 2n_i^t \ln n_i^t$, where C_t is the number of comparisons done so far and the summation gives an upper bound on the expected number of comparisons remaining.

To show that this is a supermartingale, observe that if we partition a block of size n we add n to C_t but replace the cost bound $2n \ln n$ by an expected

$$\begin{aligned} 2 \cdot \frac{1}{n}\sum_{k=0}^{n-1} 2k \ln k &\le \frac{4}{n}\int_2^n n \ln n \\ &= \frac{4}{n}\left(\frac{n^2 \ln n}{2} - \frac{n^2}{4} - \ln 2 + 1\right) \\ &= 2n \ln n - n - \ln 2 + 1 \qquad \le 2n \ln n - n. \end{aligned}$$

The net change is at most $-\ln 2$. The fact that it's not zero suggests that we could improve the $2n \ln n$ bound slightly, but since it's going down, we have a supermartingale.

Let's try to get a bound on how much X_t changes at each step. The C_t part goes up by at most $n-1$. The summation can only go down; if we split a block of size n_i, the biggest drop we get is if we split it

evenly,[9] This gives a drop of

$$\begin{aligned}
2n\ln n - 2(2\frac{n-1}{2}\ln\frac{n-1}{2}) &= 2n\ln n - 2(n-1)\ln\left(n\frac{n-1}{2n}\right) \\
&= 2n\ln n - 2(n-1)(\ln n - \ln\frac{2n}{n-1}) \\
&= 2n\ln n - 2n\ln n + 2n\ln\frac{2n}{n-1} + 2\ln n - 2\ln\frac{2n}{n-1} \\
&= 2n\cdot O(1) + O(\log n) \\
&= O(n).
\end{aligned}$$

(with a constant tending to 2 in the limit).

So we can apply (3.2.12) with $c_t = O(n)$ to the at most n steps of the algorithm, and get

$$\Pr\left[C_n - 2n\ln n \geq t\right] \leq e^{-t^2/O(n^3)}.$$

This gives $C_n = O(n^{3/2})$ with constant probability or $O(n^{3/2}\sqrt{\log n})$ with all but polynomial probability. This is a rather terrible bound, but it's a lot better than $O(n^2)$. For a much better bound, see [MH92].

3.3 Anti-concentration bounds

It may be that for some problem you want to show that a sum of random variables is far from its mean at least some of the time: this would be an **anti-concentration bound**. Anti-concentration bounds are much less well-understood than concentration bounds, but there are some results that help in some cases.

For variables where we know the distribution of the sum exactly (e.g, sums with binomial distributions, or sums we can attack with generating functions), we don't need these. But they may be useful if computing the distribution of the sum directly is hard.

3.3.1 The Berry-Esseen theorem

The **Berry-Esseen theorem**[10] characterizes how quickly a sum of identical independent random variables converges to a normal distribution, as a

[9] This can be proven most easily using convexity of $n\ln n$.

[10] Sometimes written *Berry-Esséen theorem* to help with the pronunciation of Esseen's last name.

function of the **third moment** of the random variables. Its simplest version says that if we have n independent, identically-distributed random variables $X_1 \dots X_n$, with $\mathrm{E}[X_i] = 0$, $\mathrm{Var}[X_i] = \mathrm{E}[X_i^2] = \sigma^2$, and $\mathrm{E}[|X_i|^3] \leq \rho$. Then

$$\sup_{-\infty<x<\infty} \left| \Pr\left[\frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_i \leq x \right] - \Phi(x) \right| \leq \frac{C\rho}{\sigma^3 \sqrt{n}}, \tag{3.3.1}$$

were C is an absolute constant and Φ is the normal distribution function. Note that the σ^3 in the denominator is really $\mathrm{Var}[X_i]^{3/2}$.

A classic proof of this result with $C = 3$ can be found in [Fel71, §XVI.5]. The page `http://en.wikipedia.org/wiki/Berry_Esseen_theorem` has an extensive summary of subsequent work on improving the constant; as of 2011, the best constant seems to be $0.4784 < 1/2$, from a paper by Korolev and Shevtsova [KS10].[11]

As with most results involving sums of random variables, there are generalizations to martingales. These are too involved to describe here, but see [HH80, §3.6].

3.3.2 The Littleword-Offord problem

The **Littlewood-Offord problem** asks, given a set of n complex numbers $x_1 \dots x_n$ with $|x_i| \geq 1$, for how many assignments of ± 1 to coefficients $\epsilon_1 \dots \epsilon_n$ it holds that $|\sum_{i=1}^{n} \epsilon_i x_i| \leq r$. Paul Erdős showed [Erd45] that this quantity was at most $cr2^n/\sqrt{n}$, where c is a constant. The quantity $c2^n/\sqrt{n}$ here is really $\frac{1}{2}\binom{n}{\lfloor n/2 \rfloor}$: Erdős's proof shows that for each interval of length $2r$, the number of assignments that give a sum in the interior of the interval is bounded by at most the sum of the r largest binomial coefficients.

In random-variable terms, this means that if we are looking at $\sum_{i=1}^{n} \epsilon x_i$, where the x_i are constants with $|x_i| \geq 1$ and the ϵ_i are independent ± 1 fair coin-flips, then $\Pr\left[|\sum_{i=1}^{n} \epsilon_i x_i| \leq r\right]$ is maximized by making all the x_i equal to 1. This shows that any distribution where the x_i are all reasonably large will not be any more concentrated than a binomial distribution.

There has been a lot of recent work on variants of the Littlewood-Offord problem, much of it by Terry Tao and Van Vu. See `http://terrytao.wordpress.com/2009/02/16/a-sharp-inverse-littlewood-offord-theorem/` for a summary of much of this work.

[11] Also available as `http://arxiv.org/abs/0912.2795`.

Chapter 4

The probabilistic method

In this chapter, we'll discuss the **probabilistic method**, a tool for proving the existence of objects with particular combinatorial properties. The relevance of this to randomized algorithms is that in some cases we can turn such a proof into an algorithm for *producing* objects with the desired properties.

We'll mostly be following Chapter 5 of [MR95] with some updates for more recent results. If you'd like to read more about these technique, the standard reference on the probabilistic method in combinatorics is the text of Alon and Spencer [AS92].

4.1 Basic idea

Suppose we want to show that some object exists, but we don't know how to construct it explicitly. One way to do this is to devise some random process for generating objects, and show that the probability that it generates the object we want is greater than zero. This implies that something we want exists, because otherwise it would be impossible to generate; and it works even if the nonzero probability is very, very small. The systematic development of the method is generally attributed to the notoriously productive mathematician Paul Erdős and his frequent collaborator Alfréd Rényi.

From an algorithmic perspective, the probabilistic method is useful mainly when we can make the nonzero probability substantially larger than zero—and especially if we can recognize when we've won. But sometimes just demonstrating the existence of an object is a start.

We give a couple of example of the probabilistic method in action below. In each case, the probability that we get a good outcome is actually pretty

high, so we could in principle generate a good outcome by retrying our random process until it works. There are some more complicated examples of the method for which this doesn't work, either because the probability of success is vanishingly small, or because we can't efficiently test whether what we did succeeded (the last example below may fall into this category). This means that we often end up with objects whose existence we can demonstrate even though we can't actually point to any examples of them. For example, it is known that there exist **sorting networks** (a special class of circuits for sorting numbers in parallel) that sort in time $O(\log n)$, where n is the number of values being sorted [AKS83]; these can be generated randomly with nonzero probability. But the best explicit constructions of such networks take time $\Theta(\log^2 n)$, and the question of how to find an *explicit* network that achieves $O(\log n)$ time has been open for over 25 years now despite many efforts to solve it.

4.1.1 Unique hats

A collection of n robots each wishes to own a unique hat. Unfortunately, the State Ministry for Hat Production only supplies one kind of hat. A robot will only be happy if (a) it has a hat, and (b) no robot it sees has a hat. Fortunately each robot can only see k other robots. How many robots can be happy?

We could try to be clever about answering this question, or we could apply the probabilistic method. Suppose we give each robot a hat with independent probability p. Then the probability that any particular robot r is happy is pq^k, where $q = 1 - p$ is the probability that a robot doesn't have a hat and q^k gives the probability that none of the robots that r sees has a hat. If we let X_r be the indicator for the event that r is happy, we have $\mathrm{E}[X_r] = pq^k$ and the expected number of happy robots is $\mathrm{E}[\sum X_r] = \sum \mathrm{E}[X_r] = npq^k$. Since we can achieve this value on average, there must be some specific assignment that achieves it as well.

To choose a good p, we apply the usual calculus trick of looking for a maximum by looking for the place where the derivative is zero. We have $\frac{d}{dp} npq^k = nq^k - nkpq^{k-1}$ (since $\frac{dq}{dp} = -1$), so we get $nq^k - nkpq^{k-1} = 0$. Factoring out n and q^{k-1} gives $q - pk = 0$ or $1 - p - pk = 0$ or $p = 1/(k+1)$. For this value of p, the expected number of happy robots is exactly $n\left(\frac{1}{k+1}\right)\left(1 - \frac{1}{k+1}\right)^k$. For large k the last factor approaches $1/e$, giving us approximately $(1/e)\frac{n}{k+1}$ happy robots.

Up to constant factors this is about as good an outcome as we can hope

for: we can set things up so that our robots are arranged in groups of $k+1$, where each robot sees all the other robots in its group, and here we can clearly only have 1 happy robot per group, for a maximum of $n/(k+1)$ happy robots.

Note that we can improve the constant $1/e$ slightly by being smarter than just handing out hats at random. Starting with our original n robots, look for a robot that is observed by the smallest number of other robots. Since there are only nk pairs of robots (r_1, r_2) where r_1 sees r_2, one of the n robots is only seen by at most k other robots. Now give this robot a hat, and give no hat to any robot that sees it or that it sees. This produces 1 happy robot while knocking out at most $2k+1$ robots total. Now remove these robots from consideration and repeat the analysis on the remaining robots, to get another happy robot while again knocking out at most $2k+1$ robots. Iterating this procedure until no robots are left gives at least $n/(2k+1)$ happy robots; this is close to $(1/2)\frac{n}{k+1}$ for large k, which is a bit better than $(1/e)\frac{n}{k+1}$. (It may be possible to improve the constant further with more analysis.) This shows that the probabilistic method doesn't necessarily produce better results than we can get by being smart. But the hope is that with the probabilistic method we don't have to be smart, and for some problems it seems that solving them without using the probabilistic method requires being exceptionally smart.

4.1.2 Large cuts

We've previously seen (§1.3) a randomized algorithm for finding small cuts in a graph. What if we want to find a large cut?

Here is a particularly brainless algorithm that does it. For each vertex, flip a coin: if the coin comes up heads, put the vertex in S, otherwise, put in it T. For each edge, there is a probability of exactly $1/2$ that it is included in the S–T cut. It follows that the expected size of the cut is exactly $m/2$.

One consequence of this is that every graph has a global cut that includes at least half the edges. Another is that this algorithm finds such a cut, with probability at least $\frac{1}{m+1}$ (exercise: why?). If we want to be cute we can even argue that we have a polynomial-time randomized algorithm for approximating the **maximum cut** within a factor of 2, which is pretty good considering that MAX-CUT is **NP**-hard.

4.1.3 Ramsey numbers

Consider a collection of n schoolchildren, and imagine that each pair of schoolchildren either like each other or hate each other. We assume that these preferences are symmetric: if x likes y, then y likes x, and similarly if x hates y, y hates x. Let $R(k,h)$ be the smallest value for n that ensures that among any group of n schoolchildren, there is either a subset of k children that all like each other or a subset of h children that all hate each other.[1]

It is not hard to show that $R(k,h)$ is finite for all k and h (see `http://en.wikipedia.org/wiki/Ramsey's_Theorem`). The exact value of $R(k,h)$ is known only for small values of k and h. But we can use the probabilistic method to show that for $k = h$, it is fairly large.

Theorem 4.1.1. *If $k \geq 3$, then $R(k,k) > 2^{k/2}$.*

Proof. Suppose each pair of schoolchildren flip a fair coin to decide whether they like each other or not. Then the probability that any particular set of k schoolchildren all like each other is $2^{-\binom{k}{2}}$ and the probability that they all hate each other is the same. Summing over both possibilities and all subsets gives a bound of $\binom{n}{k}2^{1-\binom{k}{2}}$ on the probability that there is at least one subset in which everybody likes or everybody hates everybody. For $n = 2^{k/2}$, we have

$$\begin{aligned}\binom{n}{k}2^{1-\binom{k}{2}} &\leq \frac{n^k}{k!}2^{1-\binom{k}{2}} \\ &= \frac{2^{k^2/2+1-k(k-1)/2}}{k!} \\ &= \frac{2^{k^2/2+1-k^2/2+k/2}}{k!} \\ &= \frac{2^{1+k/2}}{k!} \\ &< 1.\end{aligned}$$

Because the probability that there is an all-liking or all-hating subset is less than 1, there must be some chance that we get a collection that doesn't have one. So such a collection exists. It follows that $R(k,k) > 2^{k/2}$, because we have shown that not all collections at $n = 2^{k/2}$ have the Ramsey property. □

[1] In terms of graphs, any graph G with at least $R(k,h)$ nodes contains either a **clique** of size k or an **anti-clique** of size h.

The last step in the proof uses the fact that $2^{1+k/2} < k!$ for $k \geq 3$, which can be tested explicitly for $k = 3$ and proved by induction for larger k. The resulting bound is a little bit weaker than just saying that n must be large enough that $\binom{n}{k} 2^{1-\binom{k}{2}} \geq 1$, but it's easier to use.

The proof can be generalized to the case where $k \neq h$ by tweaking the bounds and probabilities appropriately. Note that even though this process generates a graph with no large cliques or anticliques with reasonably high probability, we don't have any good way of testing the result, since testing for the existence of a clique is **NP**-hard.

4.1.4 Cycles in tournaments

In the previous example we used the probabilistic method to show that there existed a structure that didn't contain some particular substructure. For this example we will do the reverse: show that there exists a structure that contains many instances of a particular substructure.

Imagine that n wrestlers wrestler a round-robin tournament, so that ever wrestler wrestles every other wrestler exactly once, and so that for each pair of wrestlers one wrestler beats the other. Consider a cycle of wrestlers $x_1, x_2, \ldots, x_n$ such that each wrestler x_i beats x_{i+1} and x_n beats x_1.

Theorem 4.1.2. *There is a possible outcome to this tournament with at least $n!2^{-n}$ such cycles.*

Proof. Flip a coin to decide the outcome of each pairing. For any particular sequence of wrestlers $x_1, \ldots, x_n$, the probability that it is a cycle is then 2^{-n} (since there are n bouts in the cycle and each has independent probability 2^{-1} of going the right way). Now let π range over all permutations of the wrestlers and define X_π as the indicator variable that permutation π gives a cycle. We've just shown that $\mathrm{E}[X_\pi] = 2^{-n}$; so the expected total number of cycles is $\mathrm{E}[\sum_\pi X_\pi] = \sum_\pi \mathrm{E}[X_\pi] = n!2^{-n}$. But if the average value is at least $n!2^{-n}$, there must be some specific outcome that gives a value at least this high. □

4.2 Applications to MAX-SAT

Like MAX-CUT, MAX-SAT is an **NP**-hard optimization problem that sounds like a very short story about Max. We are given a **satisfiability problem** in the form of a set of m **clauses**, each of which is the OR of a bunch of variables or their negations. We want to choose values for the n variables

that **satisfy** as many of the clauses as possible, where a clause is satisfied if it contains a true variable or the negation of a false variable.[2]

We can instantly satisfy at least $m/2$ clauses on average by assigning values to the variables independently and uniformly at random; the analysis is the same as in Section 4.1.2 for large cuts, since a random assignment makes the first variable in a clause true with probability $1/2$. Since this approach doesn't require thinking and doesn't use the fact that many of our clauses may have more than one variable, we can probably do better. Except we can't do better in the worst case, because it might be that our clauses consist entirely of x and $\neg x$ for some variable x; clearly, we can only satisfy half of these. We could do better if we knew all of our clauses consisted of at least k distinct literals (satisfied with probability $1 - 2^{-k}$), but we can't count on this.

Our goal then will be to get a good **approximation ratio**, defined as the ratio between the number of clauses we manage to satisfy and the actual maximum that can be satisfied. The tricky part in designing a **approximation algorithms** is showing that the denominator here won't be too big. We can do this using a standard trick, of expressing our original problem as an **integer program** and then **relaxing** it to a **linear program**[3] whose solution doesn't have to consist of integers. We then convert the fractional solution back to an integer solution by rounding some of the variables to integer values randomly in a way that preserves their expectations, a technique known as **randomized rounding**.[4]

Here is the integer program (taken from [MR95, §5.2]). We let $z_j \in \{0, 1\}$ represent whether clause C_j is satisfied, and let $y_i \in \{0, 1\}$ be the value of variable x_i. We also let C_j^+ and C_j^- be the set of variables that appear in

[2]The presentation here follows Section 5.2 of [MR95], which in turn is mostly based on a classic paper of Goemans and Williamson [GW94].

[3]A **linear program** is an optimization problem where we want to maximize (or minimize) some linear **objective function** of the variables subject to linear-inequality constraints. A simple example would be to maximize $x + y$ subject to $2x + y \leq 1$ and $x + 3y \leq 1$; here the **optimal solution** is the assignment $x = \frac{2}{5}, y = \frac{1}{5}$, which sets the objective function to its maximum possible value $\frac{3}{5}$. An **integer program** is a linear program where some of the variables are restricted to be integers. Determining if an integer program even has a solution is **NP**-complete; in contrast, linear programs can be solved in polynomial time. We can **relax** an integer program to a linear program by dropping the requirements that variables be integers; this will let us find a **fractional solution** that is at least as good as the best integer solution, but might be undesirable because it tells us to do something that is ludicrous in the context of our original problem, like only putting half a passenger on the next plane.

[4] Randomized rounding was invented by Raghavan and Thompson [RT87]; the particular application here is due to Goemans and Williamson [GW94].

C_j with and without negation. The problem is to maximize

$$\sum_{j=1}^{m} z_j$$

subject to

$$\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq z_j,$$

for all j.

The main trick here is to encode OR in the constraints; there is no requirement that z_j is the OR of the y_i and $(1-y_i)$ values, but we maximize the objective function by setting it that way.

Sadly, solving integer programs like the above is **NP**-hard (which is not surprising, since if we could solve this particular one, we could solve SAT). But if we drop the requirement that $y_i, z_j \in \{0, 1\}$ and replace it with $0 \leq y_i \leq 1$ and $0 \leq z_j \leq 1$, we get a linear program—solvable in polynomial time—with an optimal value at least as good as the value for the integer program, for the simple reason that any solution to the integer program is also a solution to the linear program.

The problem now is that the solution to the linear program is likely to be **fractional**: instead of getting useful 0–1 values, we might find out we are supposed to make x_i only 2/3 true. So we need one more trick to turn the fractional values back into integers. This is the randomized rounding step: given a fractional assignment $\hat{y}_i$, we set x_i to true with probability $\hat{y}_i$.

So what does randomized rounding do to clauses? In our fractional solution, a clause might have value $\hat{z}_j$, obtained by summing up bits and pieces of partially-true variables. We'd like to argue that the rounded version gives a similar probability that C_j is satisfied.

Suppose C_j has k variables; to make things simpler, we'll pretend that C_j is exactly $x_1 \vee x_2 \vee \ldots x_k$. Then the probability that C_j is satisfied is exactly $1 - \prod_{i=1}^{k}(1 - \hat{y}_i)$. This quantity is minimized subject to $\sum_{i=1}^{k} \hat{y}_i \geq \hat{z}_j$ by setting all $\hat{y}_i$ equal to $\hat{z}_j/k$ (easy application of Lagrange multipliers, or

can be shown using a convexity argument). Writing z for $\hat{z}_j$, this gives

$$\begin{aligned}\Pr[C_j \text{ is satisfied}] &= 1 - \prod_{i=1}^{k}(1-\hat{y}_i) \\ &\geq 1 - \prod i = 1^k (1 - z/k) \\ &= 1 - (1-z/k)^k \\ &\geq z(1-(1-1/k)^k). \\ &\geq z(1-1/e).\end{aligned}$$

The second-to-last step looks like a typo, but it actually works. The idea is to observe that the function $f(z) = 1 - (1-z/k)^k$ is concave (Proof: $\frac{d^2}{dz^2} f(z) = -\frac{k-1}{k}(1-z/k)^{k-2} < 0$), while $g(z) = z(1-(1-1/k)^k)$ is linear, so since $f(0) = 0 = g(0)$ and $f(1) = 1 - (1-1/k)^k = g(1)$, any point in between must have $f(z) \geq g(z)$.

Since each clause is satisfied with probability at least $\hat{z}_j(1-1/e)$, the expected number of satisfied clauses is at least $(1-1/e)\sum_j \hat{z}_j$, which is at least $(1-1/e)$ times the optimum. This gives an approximation ratio of slightly more than 0.632, which is better than 1/2, but still kind of weak.

So now we apply the second trick from [GW94]: we'll observe that, on a per-clause basis, we have a randomized rounding algorithm that is good at satisfying small clauses (the coefficient $(1-(1-1/k)^k)$ goes all the way up to 1 when $k = 1$), and our earlier dumb algorithm that is good at satisfying big clauses. We can't combine these directly (the two algorithms demand different assignments), but we can run both in parallel and take whichever gives a better result.

To show that this works, let X_j be indicator for the event that clause j is satisfied by the randomized-rounding algorithm and Y_j the indicator for the even that it is satisfied by the simple algorithm. Then if C_j has k literals,

$$\begin{aligned}&\mathrm{E}[X_j] + \mathrm{E}[Y_j] \\ &\qquad \geq (1-2^{-k}) + (1-(1-1/k)^k)\hat{z}_j \\ &\qquad \geq ((1-2^{-k}) + (1-(1-1/k)^k))\hat{z}_j \\ &\qquad = (2-2^{-k}-(1-1/k)^k)\hat{z}_j.\end{aligned}$$

The coefficient here is exactly 3/2 when $k = 1$ or $k = 2$, and rises thereafter, so for integer k we have $\mathrm{E}[X_j] + \mathrm{E}[Y_j] \geq (3/2)\hat{z}_j$. Summing over all j then gives $\mathrm{E}\left[\sum_j X_j\right] + \mathrm{E}\left[\sum_j Y_j\right] \geq (3/2)\sum_j \hat{z}_j$. But then one of the

two expected sums must beat $(3/4)\sum_j \hat{z}_j$, giving us a (3/4)-approximation algorithm.

4.3 The Lovász Local Lemma

Suppose we have a finite set of bad events $\mathcal{A}$, and we want to show that with nonzero probability, none of these events occur. Formally, we want to show $\Pr\left[\bigcap_{A\in\mathcal{A}} \overline{A}\right] > 0$.

Our usual trick so far has been to use the union bound (3.1.3) to show that $\sum_{A\in\mathcal{A}} \Pr[A] < 1$. But this only works if the events are actually improbable. If the union bound doesn't work, we might be lucky enough to have the events be independent; in this case, $\Pr\left[\bigcap_{A\in\mathcal{A}} \overline{A}\right] = \prod_{A\in\mathcal{A}} \Pr[\overline{A}] > 0$, as long as each event $\overline{A}$ occurs with positive probability. But most of the time, we'll find that the events we care about aren't independent, so this won't work either.

The **Lovász Local Lemma** [EL75] handles a situation intermediate between these two extremes, where events are generally not independent of each other, but each collection of events that are not independent of some particular event A has low total probability. In the original version; it's non-constructive: the lemma shows a nonzero probability that none of the events occur, but this probability may be very small if we try to sample the events at random and there is no guidance for how to find a particular outcome that makes all the events false.

Subsequent work [Bec91, Alo91, MR98, CS00, Sri08, Mos09, MT10] showed how, when the events A are determined by some underlying set of independent variables and independence between two events is detected by having non-overlapping sets of underlying variables, an actual solution could be found in polynomial expected time. The final version of this work, due to Moser and Tardos [MT10], gives the same bounds as in the original non-constructive lemma, using the simplest algorithm imaginable: whenever some bad event A occurs, squish it by resampling all of its variables, and continue until no bad events are left.

4.3.1 General version

A formal statement of the general lemma is:[5]

Lemma 4.3.1. *Let $\mathcal{A} = A_1, \ldots, A_m$ be a finite collection of events on some probability space, and for each $A \in \mathcal{A}$, let $\Gamma(A)$ be a set of events such that*

[5] This version is adapted from [MT10].

A is independent of all events not in $\Gamma^+ A = \{A\} \cup \Gamma(A)$. If there exist real numbers $x(A) \in (0,1)$ such that, for all events $A \in \mathcal{A}$

$$\Pr[A] \leq x(A) \left(\prod_{B \in \Gamma(A)} (1 - x(B)) \right), \tag{4.3.1}$$

then

$$\Pr\left[\bigcap_{A \in \mathcal{A}} \overline{A} \right] \geq \prod_{A \in \mathcal{A}} (1 - x(A)). \tag{4.3.2}$$

In particular, this means that the probability that none of the A_i occur is not zero, since we have assumed $x(A_i) < 1$ holds for all i.

The role of $x(A)$ in the original proof is to act as an upper bound on the probability that A occurs given that some collection of other events doesn't occur. For the constructive proof, the $x(A)$ are used to show a bound on the number of resampling steps needed until none of the A occur.

4.3.2 Symmetric version

For many applications, having to come up with the $x(A)$ values can be awkward. The following symmetric version is often used instead:

Corollary 4.3.2. *Let $\mathcal{A}$ and Γ be as in Lemma 4.3.1. Suppose that there are constants p and d, such that for all $A \in \mathcal{A}$, we have $\Pr[A] \leq p$ and $|\Gamma(A)| \leq d$. Then if $ep(d+1) < 1$, $\Pr\left[\bigcap_{A \in \mathcal{A}} \overline{A}\right] \neq 0$.*

Proof. Basically, we are going to pick a single value x such that $x(A) = x$ for all A in $\mathcal{A}$, and (4.3.1) is satisfied. This works as long as $p \leq x(1-x)^d$, as in this case we have, for all A, $\Pr[A] \leq p \leq x(1-x)^d \leq x(1-x)^{|\Gamma(A)|} = x(A)\left(\prod_{B \in \Gamma(A)}(1 - x(B))\right)$.

For fixed d, $x(1-x)^d$ is maximized using the usual trick: $\frac{d}{dx} x(1-x)^d = (1-x)^d - xd(1-x)^{d-1} = 0$ gives $(1-x) - xd = 0$ or $x = \frac{1}{d+1}$. So now we need $p \leq \frac{1}{d+1}\left(1 - \frac{1}{d+1}\right)^d$. It is possible to show that $1/e < \left(1 - \frac{1}{d+1}\right)^d$ for all $d \geq 0$ (see the solution to Problem A.1.4). So $ep(d+1) \leq 1$ implies $p \leq \frac{1}{e(d+1)} \leq \left(1 - \frac{1}{d+1}\right)^d)\frac{1}{d+1} \leq x(1-x)^{|\Gamma(A)|}$ as required by Lemma 4.3.1. □

4.3.3 Applications

4.3.3.1 Graph coloring

Let's start with a simple application of the local lemma where we know what the right answer should be. Suppose we want to color the vertices of a cycle with c colors, so that no edge has two endpoints with the same color. How many colors do we need?

Using brains, we can quickly figure out that $c = 3$ is enough. Without brains, we could try coloring the vertices randomly: but in a cycle with n vertices and n edges, on average n/c of the edges will be monochromatic, since each edge is monochromatic with probability $1/c$. If these bad events were independent, we could argue that there was a $(1-1/c)^n > 0$ probability that none of them occurred, but they aren't, so we can't. Instead, we'll use the local lemma.

The set of bad events $\mathcal{A}$ is just the set of events $A_i =$ [edge i is monochromatic]. We've already computed $p = 1/c$. To get d, notice that each edge only shares a vertex with two other edges, so $|\Gamma(A_i)| \leq 2$. Corollary 4.3.2 then says that there is a good coloring as long as $ep(d+1) = 3e/c \leq 1$, which holds as long as $c \geq 9$. We've just shown we can 9-color a cycle. If we look more closely at the proof of Corollary 4.3.2, we can see that $p \leq \frac{1}{3}\left(1 - \frac{1}{3}\right)^2 = \frac{4}{27}$ would also work; this says we can 7-color a cycle. Still not as good as what we can do if we are paying attention, but not bad for a procedure that doesn't use the structure of the problem much.

4.3.3.2 Satisfiability of k-CNF formulas

A more sophisticated application is demonstrating satisfiability for k-CNF formulas where each variable appears in a bounded number of clauses. Recall that a **k-CNF formula** consists of m **clauses**, each of which consists of exactly k variables or their negations (collectively called **literals**. It is **satisfied** if at least one literal in every clause is assigned the value true.

Suppose that each variable appears in at most ℓ clauses. Let $\mathcal{A}$ consist of all the events $A_i =$ [clause i is not satisfied]. Then, for all i, $\Pr[A_i] = 2^{-k}$ exactly, and $|\Gamma A_i| \leq d = k(\ell - 1)$ since each variable in A_i is shared with at most $\ell - 1$ other clauses, and A_i will be independent of all events A_j with which it doesn't share a variable. So if $ep(d+1) = e2^{-k}k(\ell-1) \leq 1$, or equivalently if $\ell \leq \frac{2^k}{ek} + 1$, Corollary 4.3.2 tells us that a satisfying assignment exists.[6]

[6] To avoid over-advertising this claim, it's worth noting that the bound on ℓ only reaches 2 at $k = 4$, although it starts growing pretty fast after that.

Corollary 4.3.2 doesn't let us actually find a satisfying assignment, but it turns out we can do that too. We'll return to this when we talk about the constructive version of the local lemma in Section 4.3.5

4.3.4 Non-constructive proof

This is essentially the same argument presented in [MR95, §5.5], but we adapt the notation to match the statement in terms of neighborhoods $\Gamma(A)$ instead of edges in a **dependency graph**.

We show by induction on $|S|$ that for any A and any $S \subseteq \mathcal{A}$ with $A \notin S$,

$$\Pr\left[A \middle| \bigcap_{B \in S} \overline{B}\right] \leq x(A). \tag{4.3.3}$$

When $|S| = 0$, this just says $\Pr[A] \leq x(A)$, which follows immediately from (4.3.1).

For larger S, split S into $S_1 = S \cap \Gamma(A)$, the events in S that might not be independent of A; and $S_2 = S \setminus \Gamma(A)$, the events in S that we know to be independent of A. Write C_1 for the event $\bigcap_{B \in S_1} \overline{B}$ and C_2 for the event $\bigcap_{B \in S_2} \overline{B}$. Then

$$\begin{aligned}
\Pr\left[A \middle| \bigcap_{B \in S} \overline{B}\right] &= \frac{\Pr[A \cap C_1 \cap C_2]}{\Pr[C_1 \cap C_2]} \\
&= \frac{\Pr[A \cap C_1 | C_2] \Pr[C_2]}{\Pr[C_1 | C_2] \Pr[C_2]} \\
&= \frac{\Pr[A \cap C_1 | C_2]}{\Pr[C_1 | C_2]}.
\end{aligned} \tag{4.3.4}$$

We don't need to do anything particularly clever with the numerator:

$$\begin{aligned}
\Pr[A \cap C_1 | C_2] &\leq \Pr[A | C_2] \\
&= \Pr[A] \\
&\leq x(A) \left(\prod_{B \in \Gamma(A)} (1 - x(B)) \right),
\end{aligned} \tag{4.3.5}$$

from (4.3.1) and the fact that A is independent of all B in S_2 and thus of C_2.

For the denominator, we expand C_1 back out to $\bigcap_{B \in S_1} \overline{B}$ and break out the induction hypothesis. To bound $\Pr\left[\bigcap_{B \in S_1} \overline{B} \middle| C_2\right]$, we order S_1

arbitrarily as $\{B_1, \ldots, B_r\}$ for some r and show by induction on ℓ as ℓ goes from 1 to r that

$$\Pr\left[\bigcap_{i=1}^{\ell} \overline{B}_i \middle| C_2\right] \geq \prod_{i=1}^{\ell}(1 - x(B_i)). \tag{4.3.6}$$

The proof is that, for $\ell = 1$,

$$\begin{aligned}\Pr\left[\overline{B}_1 \middle| C_2\right] &= 1 - \Pr\left[B_1 \middle| C_2\right] \\ &\geq 1 - x(B_1)\end{aligned}$$

using the outer induction hypothesis (4.3.3), and for larger ℓ, we can compute

$$\begin{aligned}\Pr\left[\bigcap_{i=1}^{\ell} \overline{B}_i \middle| C_2\right] &= \Pr\left[\overline{B}_\ell \middle| \left(\bigcap_{i=1}^{\ell-1} \overline{B}_i\right) \cap C_2\right] \cdot \Pr\left[\bigcap_{i=1}^{\ell-1} \overline{B}_i \middle| C_2\right] \\ &\geq (1 - x(B_\ell)) \prod_{i=1}^{\ell-1}(1 - x(B_i)) \\ &= \prod_{i=1}^{\ell}(1 - x(B_i)),\end{aligned}$$

where the second-to-last step uses the outer induction hypothesis (4.3.3) for the first term and the inner induction hypothesis (4.3.6) for the rest. This completes the proof of the inner induction.

When $\ell = r$, we get

$$\begin{aligned}\Pr[C_1|C_2] &= \Pr\left[\bigcap_{i=1}^{r} \overline{B}_i \middle| C_2\right] \\ &\geq \prod_{B \in S_1} (1 - x(B)).\end{aligned} \tag{4.3.7}$$

Substituting (4.3.5) and (4.3.7) into (4.3.4) gives

$$\begin{aligned}\Pr\left[A \middle| \bigcap_{B \in S} \overline{B}\right] &\leq \frac{x(A)\left(\prod_{B \in \Gamma(A)}(1 - x(B))\right)}{\prod_{B \in S_1}(1 - x(B))} \\ &= x(A)\left(\prod_{B \in \Gamma(A) \setminus S_1} (1 - x(B))\right) \\ &\leq x(A).\end{aligned}$$

This completes the proof of the outer induction.

To get the bound (4.3.2), we reach back inside the proof and repeat the argument for (4.3.7) with $\bigcap_{A\in\mathcal{A}} \overline{A}$ in place of C_1 and without the conditioning on C_2. We order $\mathcal{A}$ arbitrarily as $\{A_1, A_2, \ldots, A_m\}$ and show by induction on k that

$$\Pr[\left[\bigcap_{i=1}^{k} \overline{A}_i\right] \geq \prod_{i=1}^{k}(1 - x(A_k)). \tag{4.3.8}$$

For the base case we have $k = 0$ and $\Pr[\Omega] \geq 1$, using the usual conventions on empty products. For larger k, we have

$$\begin{aligned}\Pr[\left[\bigcap_{i=1}^{k} \overline{A}_i\right] &= \Pr\left[\overline{A}_k \middle| \bigcap_{i=1}^{k-1} \overline{A}_i\right] \\ &\geq (1 - x(A_k)) \prod_{i=1}^{k-1}(1 - x(A_i)) \\ &\geq \prod_{i=1}^{k}(1 - x(A_k)),\end{aligned}$$

where in the second-to-last step we use (4.3.3) for the first term and (4.3.8) for the big product.

Setting $k = n$ finishes the proof.

4.3.5 Constructive proof

We now describe he constructive proof of the Lovász local lemma due to Moser and Tardos [MT10], which is based on a slightly more specialized construction of Moser alone [Mos09]. This version applies when our set of bad events $\mathcal{A}$ is defined in terms of a set of *independent* variables $\mathcal{P}$, where each $A \in \mathcal{A}$ is determined by some set of variables $\text{vbl}(A) \subseteq \mathcal{P}$, and $\Gamma(A)$ is defined to be the set of all events $B \neq A$ that share at least one variable with A; i.e., $\Gamma(A) = \{B \in \mathcal{A} - \{A\} | \, \text{vbl}(B) \cap \text{vbl}(A) \neq \emptyset\}$.

In this case, we can attempt to find an assignment to the variables that makes none of the A occur using the obvious algorithm of sampling an initial state randomly, then resampling all variables in $\text{vbl}(A)$ whenever we see some bad A occur. Astonishingly, this actually works in a reasonable amount of time, without even any cleverness in deciding which A to resample at each step, if the conditions for Lemma 4.3.1 hold for x that are not too large.

In particular, we will show that a good assignment is found after each A is resampled at most $\frac{x(A)}{1-x(A)}$ times on average.

Intuitively, we might expect this to work because if each $A \in \mathcal{A}$ has a small enough neighborhood $\Gamma(A)$ and a low enough probability, then whenever we resample A's variables, it's likely that we fix A and unlikely that we break too many B in A's neighborhood. It turns out to be tricky to quantify how this process propagates outward, so the actual proof uses a different idea that essentially looks at this process in reverse, looking for each resampled event A at a set of previous events whose resampling we can blame for making A occur, and then showing that this tree (which will include every resampling operation as one of its vertices) can't be too big.

The first step is to fix some strategy for choosing which event A to resample at each step. We don't care what this strategy is; we just want it to be the case that the sequence of events depends only on the random choices made by the algorithm in its initial sampling and each resampling. We can then define an **execution log** C that lists the sequence of events $C_1, C_2, C_3, \ldots$ that are resampled at each step of the execution.

From C we now construct a **witness tree** T_t for each resampling step t whose nodes are labeled with events, with the property that the children of a node v labeled with event A_v are labeled with events in $\Gamma^+(A_v) = \{A_v\} \cap \Gamma(A_v)$. The root of T_t is labeled with C_t; to construct the rest of the tree, we work backwards through $C_{t-1}, C_{t-2}, \ldots, C_1$, and for each event C_i we encounter we attach C_i as a child of the deepest v we can find with $C_i \in \Gamma^+(A_v)$, choosing arbitrarily if there is more than one such v, and discarding C_i if there is no such v.

Now we can ask what the probability is that we see some particular witness tree τ in the execution log. Each vertex of τ corresponds to some event A_v that we resample because it occurs; in order for it to occur, the previously assignments of each variable in $\operatorname{vbl} A_v$ must have made A_v true, which occurs with probability $\Pr[A_v]$. But since we resample all the variables in A_v, any subsequent assignments to these variables are independent of the ones that contributed to v; with sufficient handwaving (or a rather detailed coupling argument as found in [MT10]) this gives that each event A_v occurs with independent probability $\Pr[A_v]$, giving $\Pr[\tau] = \prod_{v \in \tau} \Pr[A_v]$.

Why do we care about this? Because every event we resample is the root of some witness tree, and we can argue that every event we resample is the root of a *distinct* witness tree. The proof is that since we only discard events B that have $\operatorname{vbl} B$ disjoint from all nodes already in the tree, once we put A at the root, any other instance of A gets included. So the witness tree rooted at the i-th occurrence of A in C will include exactly i copies of

A, unlike the witness tree rooted at the j-th copy for $j \neq i$.

Now comes the sneaky trick: we'll count how many distinct witness trees τ we can possibly have rooted at A, given that each occurs with probability $\prod_{v \in \tau} \Pr[A_v]$. This is done by constructing a **branching process** using the $x(B)$ values from Lemma 4.3.1 as probabilities of a node with label A having a kid labeled B for each B in $\Gamma^+(A)$, and doing algebraic manipulations on the resulting probabilities until $\prod_{v \in \tau} \Pr[A_v]$ shows up.

Formally, consider the process where we construct a tree by starting with a root labeled A, and for each vertex v with label A_v, giving it a child labeled B for each $B \in \Gamma^+(A_v)$ with independent probability $x(B)$. We'll now calculate the probability p_τ that this process generates a particular tree τ.

Let $x'(B) = x(B) \prod_{C \in \Gamma(B)} (1 - x(C))$. Note that (4.3.1) says precisely that $\Pr[B] \leq x'(B)$.

For each vertex v in τ, let $W_v \subseteq \Gamma^+(A_v)$ be the set of events $B \in \Gamma^+(A_v)$ that *don't* occur as labels of children of v. The probability of getting τ is equal to the product of the probabilities at each v of getting all of its children and none of its non-children. The non-children of v collectively contribute $\prod_{B \in W_v} (1 - x(B))$ to the product, and v itself contributes $x(A_v)$ (via the product for its parent), unless v is the root node. So we can express the giant product as

$$p_\tau = \frac{1}{x(A)} \prod_{v \in \tau} \left(x(A_v) \prod_{B \in W_v} (1 - x(B)) \right).$$

We don't like the W_v very much, so we get rid of them by pushing a $(1-x(v))$ up from each vertex v, which we compensate for by dividing by $1 - x(v)$ at v itself, with an extra $1 - x(A)$ showing up at the root. This gives

$$\begin{aligned} p_\tau &= \frac{1 - x(A)}{x(A)} \prod_{v \in \tau} \left(\frac{x(A_v)}{1 - x(A_v)} \prod_{B \in \Gamma^+(A_v)} (1 - x(B)) \right). \\ &= \frac{1 - x(A)}{x(A)} \prod_{v \in \tau} \left(x(A_v) \prod_{B \in \Gamma(A_v)} (1 - x(B)) \right). \\ &= \frac{1 - x(A)}{x(A)} \prod_{v \in \tau} x'(A_v). \end{aligned}$$

Now we can bound the expected number of trees rooted at A that appear in C, assuming (4.3.1) holds. Letting $\mathcal{T}_A$ be the set of all such trees and N_A

the number that appear in C, we have

$$\begin{aligned}
\mathrm{E}[N_A] &= \sum_{\tau \in \mathcal{T}_A} \Pr[\tau \text{ appears in } C] \\
&\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in \tau} \Pr[A(v)] \\
&\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in \tau} x'(v) \\
&= \sum_{\tau \in \mathcal{T}_A} \frac{x(A)}{1 - x(A)} p_\tau \\
&= \frac{x(A)}{1 - x(A)} \sum_{\tau \in \mathcal{T}_A} p_\tau \\
&\leq \frac{x(A)}{1 - x(A)}.
\end{aligned}$$

And we're done.

Chapter 5

Derandomization

Derandomization is the process of taking a randomized algorithm and turning it into a deterministic algorithm. This is useful both for practical reasons (deterministic algorithms are more predictable, which makes them easier to debug and gives hard guarantees on running time) and theoretical reasons (if we can derandomize any randomized algorithm we could show results like $\mathbf{P} = \mathbf{RP}$, which would reduce the number of complexity classes that complexity theorists otherwise have to deal with). It may also be the case that derandomizing a randomized algorithm can be used for **probability amplification**, where we replace a low probability of success with a higher probability, in this case 1.

There are basically two approaches to derandomization:

1. Reducing the number of random bits used down to $O(\log n)$, and then searching through all choices of random bits exhaustively. For example, if we only need pairwise indepences, we could use the XOR technique from 3.2.1.1 to replace a large collection of variables with a small collection of random bits.

   Except for the exhaustive search part, this is how randomized algorithms are implemented in practice: rather than burning random bits continuously, a **pseudorandom generator** is initialized from a **seed** consisting of a small number of random bits. For pretty much all of the randomized algorithms we know about, we don't even need to use a particularly strong pseudorandom generator. This is largely because current popular generators are the products of a process of evolution: pseudorandom generators that cause wonky behavior or fail to pass tests that approximate the assumptions made about them by typical

randomized algorithms are abandoned in favor of better generators.[1]

From a theoretical perspective, pseudorandom generators offer the possibility of eliminating randomization from all randomized algorithms, except there is a complication. While (under reasonable cryptographic assumptions) there exist **cryptographically secure pseudorandom generators** whose output is indistinguishable from a genuinely random source by polynomial-time algorithms (including algorithms originally intended for other purposes), such generators are inherently incapable of reducing the number of random bits down to the $O(\log n)$ needed for exhaustive search. The reason is that any pseudorandom generator with only polynomially-many seeds can't be cryptographically secure, because we can distinguish it from a random source by just checking its output against the output for all possible seeds. Whether there is some other method for transforming an arbitrary algorithm in **RP** or **BPP** into a deterministic algorithm remains an open problem in complexity theory (and beyond the scope of this course).

2. The other approach to getting rid of randomness is to start with a specific randomized protocol and analyze its behavior enough that we can replace the random bits it uses with specific, deterministically-chosen bits we can compute. This is the main approach we will describe below. A non-constructive variant of this shows that we can always replace the random bits used by all inputs of a given size with a few carefully-selected fixed sequences (Adleman's Theorem, described in Section 5.2). More practical is the **method of conditional probabilities**, which chooses random bits sequentially based on which value is more likely to give a good outcome (see Section 5.3).

5.1 Deterministic vs. randomized algorithms

In thinking about derandomization, it can be helpful to have more than one way to look at a randomized algorithm. So far, we've describe randomized algorithms as random choices of deterministic algorithms ($M_r(x)$) or, equivalently, as deterministic algorithms that happen to have random inputs

[1] Having cheaper computers helps here as well. Nobody would have been willing to spend 2496 bytes on the state vector for Mersenne Twister [MN98] back in 1975, but in 2011 this amount of memory is trivial for pretty much any computing device except the tiniest microcontrollers.

$(M(r, x))$. This gives a very static view of how randomness is used in the algorithm. A more dynamic view is obtained by thinking of the computation of a randomized algorithm as a **computation tree**, where each path through the tree corresponds to a computation with a fixed set of random bits and a branch in the tree corresponds to a random decision. In either case we want an execution to give us the right answer with reasonably high probability, whether that probability measures our chance of getting a good deterministic machine for our particular input or landing on a good computation path.

5.2 Adleman's theorem

The idea of picking a good deterministic machine is the basis for **Adleman's theorem**[Adl78], a classic result in complexity theory. Adleman's theorem says that we can always replace randomness by an oracle that presents us with a fixed string of **advice** p_n that depends only on the size of the input n. The formal statement of the theorem relates the class **RP**, which is the class of problems for which there exists a polynomial-time Turing machine $M(x, r)$ that outputs 1 at least half the time when $x \in L$ and never when $x \notin L$; and the class **P/poly**, which is the class of problems for which there is a polynomial-sized string p_n for each input size n and a polynomial-time Turing machine M' such that $M'(x, p_{|x|})$ outputs 1 if and only if $x \in L$.

Theorem 5.2.1. $\mathbf{RP} \subseteq \mathbf{P/poly}$.

Proof. The intuition is that if any one random string has a constant probability of making M happy, then by choosing enough random strings we can make the probability that M fails using on every random string for any given input so small that even after we sum over all inputs of a particular size, the probability of failure is still small using the union bound (3.1.3). This is an example of **probability amplification**, where we repeat a randomized algorithm many times to reduce its failure probability.

Formally, consider any fixed input x of size n, and imagine running M repeatedly on this input with $n + 1$ independent sequences of random bits $r_1, r_2, \ldots, r_{n+1}$. If $x \notin L$, then $M(x, r_i)$ never outputs 1. If $x \in L$, then for each r_i, there is an independent probability of at least $1/2$ that $M(x, r_i) = 1$. So $\Pr[M(x, r_i) = 0] \leq 1/2$, and $\Pr[\forall i M(x, r_i) = 0] \leq 2^{-(n+1)}$. If we sum this probability of failure for each individual $x \in L$ of length n over the at most 2^n such elements, we get a probability that any of them fail of at most $2^n 2^{-(n+1)} = 1/2$. Turning this upside down, any sequence of $n + 1$ random

inputs includes a **witness** that $x \in L$ for *all* inputs x with probability at least 1/2. It follows that a good sequence $r_1, \ldots, r_n$, exists.

Our advice p_n is now some good sequence $p = \langle 1 \ldots r_{n+1} \rangle$, and the deterministic advice-taking algorithm that uses it runs $M(x, p_i)$ for each p_i and returns true if and only if at least one of these executions returns true. □

The classic version of this theorem shows that anything you can do with a polynomial-size randomized circuit (a circuit made up of AND, OR, and NOT gates where some of the inputs are random bits, corresponding to the r input to M) can be done with a polynomial-size deterministic circuit (where now the p_n input is baked into the circuit, since we need a different circuit for each size n anyway). This shows that ordinary algorithms are better described by **uniform** families of circuits, where there exists a polynomial-time algorithm that, given input n, outputs the circuit C_n for processing size-n inputs. The class of circuits generated by Adleman's theorem is most likely **non-uniform**: the process of finding the good witnesses p_i is not something we can clearly do in polynomial time (with the usual caveat that we can't prove much about what we can't do in polynomial time).

5.3 The method of conditional probabilities

The **method of conditional probabilities** [Rag88] follows an execution of the randomized algorithm, but at each point where we would otherwise make a random decision, makes a decision that minimizes the conditional probability of losing.

Structurally, this is similar to the method of bounded differences (see Section 3.2.3.3. Suppose our randomized algorithm generates m random values $X_1, X_2, \ldots, X_m$. Let $f(X_1, \ldots, X_m)$ be the indicator variable for our randomized algorithm failing (more generally, we can make it an expected cost or some other performance measure). Extend f to shorter sequences of values by defining $f(x_1, \ldots, x_k) = \mathrm{E}[f(x_1, \ldots, x_k, X_{k+1}, \ldots, X_m)]$. Then $Y_k = f(X_1, \ldots, X_k)$ is a Doob martingale, just as in the method of bounded differences. This implies that, for any partial sequence of values $x_1, \ldots, x_k$, there exists some next value x_{k+1} such that $f(x_1, \ldots, x_k) \geq f(x_1, \ldots, x_k, x_{k+1})$. If we can find this value, we can follow a path on which f always decreases, and obtain an outcome of the algorithm $f(x_1, \ldots, x_m)$ less than or equal to the initial value $f(\langle\rangle)$. If our outcomes are 0–1 (as in failure probabilities), and our initial value for f is less than 1, this means that we reach an outcome with $f = 0$.

The tricky part here is that it may be hard to compute $f(x_1, \ldots, x_k)$. (It's always possible to do so in principle by enumerating all assignments of the remaining variables, but if we have time to do this, we can just search for a winning assignment directly.) What makes the method of conditional probabilities practical in many cases is that we don't need f to compute the actual probability of failure, as long as f (a) gives an upper bound on the real probability of failure, at least in terminal configurations, and (b) f has the property used in the argument that for any partial sequence $x_1, \ldots, x_k$ there exists an extension $x_1, \ldots, x_k, x_{k+1}$ with $f(x_1, \ldots, x_k) \geq f(x_1, \ldots, x_k, x_{k+1}$. Such an f is called a **pessimistic estimator**. If we can find a pessimistic estimator that is easy to compute and starts less than 1, then we can just follow it down the tree to a leaf that doesn't fail.

5.3.1 A trivial example

Here is a very bad randomized algorithm for generating a string of n zeros: flip n coins, and output the results. This has a $1 - 2^{-n}$ probability of failure, which is not very good.

We can derandomize this algorithm and get rid of all probability of failure at the same time. Let $f(x_1, \ldots, x_k)$ be the probability of failure after the first k bits. Then we can easily compute f (it's 1 if any of the bits are 1 and $1 - 2^{-(n-k)}$ otherwise). We can use this to find a bit x_{k+1} at each stage that reduces f. After noticing that all of these bits are zero, we improve the deterministic algorithm even further by leaving out the computation of f.

5.3.2 Deterministic construction of Ramsey graphs

Here is an example that is slightly less trivial. For this example we let f be a count of bad events rather than a failure probability, but the same method applies.

Recall from Section 4.1.3 that if $k \geq 3$, for $n \leq 2^{k/2}$ there exists a graph with n nodes and no cliques or anticliques of size k. The proof of this fact is by observing that each subset of k vertices is bad with probability 2^{-k+1}, and when $\binom{n}{k}2^{-k+1} < 1$, the expected number of bad subsets in $G_{n,1/2}$ is less than 1, showing that some good graph exists.

We can turn this into a deterministic $n^{O(\log n)}$ algorithm for finding a Ramsey graph in the largest case when $n = 2^{k/2}$. The trick is to set the edges to be present or absent one at a time, and for each edge, take the value that minimizes the expected number of bad subsets conditioned on the choices so far. We can easily calculate the conditional probability that

a subset is bad in $O(k)$ time: if it already has both a present and missing edge, it's not bad. Otherwise, if we've already set ℓ of its edges to the same value, it's bad with probability exactly $2^{-k+\ell}$. Summing this over all $O(n^k)$ subsets takes $O(n^k k)$ time per edge, and we have $O(n^2)$ edges, giving $O(n^{k+2}k) = O\left(n^{(2\lg n+2+\lg\lg n)}\right) = n^{O(\log n)}$ time total.

It's worth mentioning that there are better deterministic constructions in the literature. The best current construction that I am aware of (as of 2011) is given by an algorithm of Barak *et al.* [BRSW06], which constructs a graph of size $k^{\log^c k}$ vertices with no clique or anticlique of size k for any fixed c.

5.3.3 Set balancing

Here we have a collection of vectors $v_1, v_2, \ldots, v_n$ in $\{0,1\}^m$. We'd like to find ± 1 coefficients $\epsilon_1, \epsilon_2, \ldots \epsilon_n$ that minimizes $\max_j |X_j|$ where $X_j = \sum_{i=1}^n \epsilon_i v_{ij}$.

If we choose the ϵ_i randomly, Hoeffding's inequality (3.2.8) says for each fixed j that $\Pr[X_j > t] < 2\exp(-t^2/2n)$ (since there are at most n non-zero values v_{ij}). The right-hand side is less than $2e/m$ for $t = \sqrt{2n(1+\ln m)}$, giving (by the union bound) a probability of less than $2/e$ that at least one of the sums exceeds $\sqrt{2n(1+\ln m)}$.

Because there may be very complicated dependence between the X_j, it is difficult to calculate the probability of the event $\cup_j[|X_j| \geq t]$, whether conditioned on some of the ϵ_i or not. However, we can calculate the probability of the individual events $[|X_j| \geq t]$ exactly. Conditioning on $\epsilon_1, \ldots, \epsilon_k$, the expected value of X_j is just $\ell = \sum_{i=1}^k \epsilon_i v_{ij}$, and the distribution of $Y = X_j - \mathrm{E}[X_j]$ is the sum of at most $n-k$ independent ± 1 random variables. The probability that $|X_j|$ exceeds t is then one minus the probability that $-t - \ell \leq Y \leq t - \ell$. We can calculate this probability exactly by summing up a linear-sized pile of binomial coefficients, which we can do in polynomial time.

For our pessimistic estimator, we take $U(\epsilon_1, \ldots, \epsilon_k) = \sum_{i=j}^n \Pr[|X_j| > t|\epsilon_1, \ldots, \epsilon_k]$. Since each term in the sum is a Doob martinagle, we have $\mathrm{E}[U(\epsilon_1, \ldots, \epsilon_{k+1}|\epsilon_1, \ldots, \epsilon_k] = U(\epsilon_1, \ldots, \epsilon_k)$, from which we immediately get that for any choice of $\epsilon_1, \ldots, \epsilon_k$ there exists some ϵ_{k+1} such that $U(\epsilon_1, \ldots, \epsilon_k) \geq U(\epsilon_1, \ldots, \epsilon_{k+1})$, and we can compute this winning ϵ_{k+1} explicitly. Furthermore, when we fix $t = \sqrt{2n\ln n} + 1$, our previous argument shows that $U(\langle\rangle) < 1$, which implies that our ultimately $U(\epsilon_1, \ldots, \epsilon_n)$ will also be less than 1; thus it's 0 and we find an assignment in which $|X_j| < \sqrt{2n\ln n} + 1$ for all j.

Chapter 6

Martingales and stopping times

In Section 3.2.3.2, we used martingales to show processes were not too wide, now we'll use them to show processes are not too long.

The general form of a martingale $\{X_t, \mathcal{F}_t\}$ consists of:

- A sequence of random variables $X_0, X_1, X_2, \ldots$; and
- A **filtration** $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \ldots$, where each σ-algebra $\mathcal{F}_t$ represents our knowledge at time t;

subject to the requirements that:

1. The sequence of random variables is **adapted** to the filtration, which just means that each X_t is measurable $\mathcal{F}_t$ or equivalently that $\mathcal{F}_t$ (and thus all subsequent $\mathcal{F}_{t'}$ for $t' \geq t$) includes all knowledge of X_t; and
2. The **martingale property** $\mathrm{E}[X_{t+1}|\mathcal{F}_t]$ holds for all t.

Given a filtration $\{\mathcal{F}_t\}$, a random variable τ is a **stopping time** for $\{\mathcal{F}_t\}$ if $\tau \geq 0$ and the event $[\tau \leq t]$ is measurable $\mathcal{F}_t$ for all t. In simple terms, τ is a stopping time if you know at time t whether to stop there or not.

6.1 The optional stopping theorem

Stopping times are very handy when used with martingales, because of the **optional stopping theorem**:

Theorem 6.1.1. *If $(X_t, \mathcal{F}_t)$ is a martingale and τ is a stopping time for $\{\mathcal{F}_t\}$, then $\mathrm{E}[X_\tau] = E[X_0]$ if*

1. $\Pr[\tau < \infty] = 1$,
2. $\mathrm{E}[|X_\tau|] < \infty$, *and*
3. $\lim_{t\to\infty} \mathrm{E}[X_t \cdot [\tau > t]] = 0$.

The first condition says that τ is finite with probability 1 (i.e., eventually we do stop). The second condition puts a bound on how big $|X_\tau|$ can get, which excludes some bad outcomes where we accept a small probability of a huge loss in order to get a large probability of a small gain. The third says that the contribution of large values of t to $\mathrm{E}[X_\tau]$ goes to zero as we consider larger and larger t; the term $[\tau > t]$ is the indicator variable for the event that τ is larger than t.

It would be nice if we could show $\mathrm{E}[X_\tau] = E[X_0]$ without the side conditions, but in general this isn't true. For example, the double-after-losing martingale strategy eventually yields $+1$ with probability 1, so if τ is the time we stop playing, we have $\Pr[\tau < \infty] = 1$, $\mathrm{E}[|X_\tau|] < \infty$, but $\mathrm{E}[X_\tau] = 1 \neq \mathrm{E}[X_0] = 0$. Here we have $\mathrm{E}[X_t \cdot [\tau > t]] = -2^{t-1}(1/2)^{t-1} = -1$ for $t > 0$, so it's the third condition that's violated. (The same thing happens for a simple ± 1 random walk that stops at $+1$, but it's a bit harder to calculate $\mathrm{E}[X_t \cdot [\tau > t]]$ in this case.)

6.2 Proof of the optional stopping theorem

To prove the optional stopping theorem, it helps to start with a simpler version:

Lemma 6.2.1. *If $(X_t, \mathcal{F}_t)$ is a martingale and τ is a stopping time for $\{\mathcal{F}_t\}$, then for any $n \in \mathbb{N}$, $\mathrm{E}[X_{\min(\tau,n)}] = E[X_0]$.*

Proof. Define $Y_t = X_0 + \sum_{i=1}^{t}(X_t - X_{t-1})[\tau > t-1]$. Then $(Y_t, \mathcal{F}_t)$ is a martingale since we can treat $[\tau \leq t-1]$ as a sequence of bets. But then $\mathrm{E}[X_{\min(\tau,n)}] = \mathrm{E}[Y_n] = \mathrm{E}[Y_0] = \mathrm{E}[X_0]$. □

So now we'll prove the full version by considering $\mathrm{E}[X_{\min(\tau,n)}]$ and showing that, under the conditions of the theorem, it approaches $\mathrm{E}[X_\tau]$ as n goes to infinity. First observe that, for any n, $X_\tau = X_{\min(\tau,n)} + [\tau > n](X_\tau - X_n)$, because either $\tau \leq n$, and we just get X_τ, or $\tau > n$, and we get $X_n + (X_\tau - X_n) = X_\tau$. Now take expectations: $\mathrm{E}[X_\tau] = \mathrm{E}[X_{\min(\tau,n)}] + \mathrm{E}[[\tau >$

$n]X_\tau] - \mathrm{E}[[\tau > n]X_n]$. Condition (3) on the theorem gives $\lim_{n\to\infty} \mathrm{E}[[\tau > n]X_n] \to 0$. If we can show that the middle term also vanishes in the limit, we are done.

Here we use condition (2). Observe that $\mathrm{E}[[\tau > n]X_\tau] = \sum_{t=n+1}^{\infty} \mathrm{E}[[\tau = t]X_t]$. Compare this with $\mathrm{E}[X_\tau] = \sum_{t=0}^{\infty} \mathrm{E}[[\tau = t]X_t]$; this is an absolutely convergent series (this is why we need condition 2), so in the limit the sum of the terms for $i = 0 \ldots n$ converges to $\mathrm{E}[X_\tau]$. But this means that the sum of the remaining terms for $i = n + 1 \ldots \infty$ converges to zero. So the middle term goes to zero as n goes to infinity.

We thus have $\mathrm{E}[X_\tau] = \lim_{n\to\infty} \mathrm{E}[X_{\min(\tau,n)}] + \mathrm{E}[[\tau > n]X_\tau] - \mathrm{E}[[\tau > n]X_n] = \lim_{n\to\infty} \mathrm{E}[X_{\min(\tau,n)}] = \mathrm{E}[X_0]$. This completes the proof.

6.3 Variants

Using the full-blown optional stopping theorem is a pain in the neck, because conditions (2) and (3) are often hard to test directly. So in practice one generally chooses one of several weaker but easier variants:

Corollary 6.3.1. *If $(X_t, \mathcal{F}_t)$ is a martingale and τ is a stopping time for $\{\mathcal{F}_t\}$ with $\tau \le n$ always for some fixed n, then $\mathrm{E}[X_\tau] = E[X_0]$.*

Proof. Here $X_\tau = X_{\min(\tau,n)}$; use Lemma 6.2.1. □

Corollary 6.3.2. *If $(X_t, \mathcal{F}_t)$ is a martingale and τ is a stopping time for $\{\mathcal{F}_t\}$ with $\Pr[\tau < \infty] = 1$ and $|X_t| \le M$ always for some fixed M and all t, then $\mathrm{E}[X_\tau] = E[X_0]$.*

Proof. Use Theorem 6.1.1, with (1) given, (2) implied by $|X_t| \le M$, and (3) following from $|\mathrm{E}[X_t[\tau > t]| \le \mathrm{E}[|X_t|\,[\tau > t]] \le M \Pr[\tau > t] \to 0$ since $\Pr[\tau < \infty] = 1$. □

Corollary 6.3.3. *If $(X_t, \mathcal{F}_t)$ is a martingale and τ is a stopping time for $\{\mathcal{F}_t\}$ where $\mathrm{E}[\tau] \le \infty$ and $|X_t - X_{t-1}| \le c$ always for some fixed c and all t, then $\mathrm{E}[X_\tau] = \mathrm{E}[X_0]$.*

Proof. Here we go back into the original proof of Theorem 6.1.1, but there is a simpler argument that $\mathrm{E}[(X_\tau - X_n)[\tau > n]]$ converges to 0. The idea is that $|E[(X_\tau - X_n)[\tau > n]]| = \left|\mathrm{E}[\sum_{t\ge n}(X_{t+1} - X_t)[\tau > t]]\right| \le \mathrm{E}[\sum_{t\ge n} |(X_{t+1} - X_t)|\,[\tau > t]] \le \mathrm{E}[\sum_{t\ge n} c[\tau > t]]$. Now use the fact that $\mathrm{E}[\tau] = \sum_t [\tau > t]$ to show that this is again the tail of a convergent sequence, and thus converges to zero. □

The short version: $\mathrm{E}[X_\tau] = \mathrm{E}[X_0]$ if at least one of the following conditions holds:

1. $\tau \leq n$ always. (Bounded time; Corollary 6.3.1.)
2. $|X_t| \leq M$ always and $\Pr[\tau < \infty] = 1$. (Bounded range with finite time; Corollary 6.3.2.)
3. $|X_t - X_{t-1}| \leq c$ and $\mathrm{E}[\tau] < \infty$. (Bounded increments with finite expected time; Corollary 6.3.3.)

6.4 Applications

6.4.1 Random walks

Let X_t be an **unbiased** ± 1 **random walk** that starts at 0, adds ± 1 to its current position with equal probability at each step, and stops if it reaches $+a$ or $-b$.[1] We'd like to calculate the probability of reaching $+a$ before $-b$. Let τ be the time at which the process stops. We can easily show that $\Pr[\tau < \infty] = 1$ and $\mathrm{E}[\tau] < \infty$ by observing that from any state of the random walk, there is a probability of at least $2^{-(a+b)}$ that it stops within $a + b$ steps (by flipping heads $a + b$ times in a row), so that if we consider a sequence of intervals of length $a+b$, the expected number of such intervals we can have before we stop is at most 2^{a+b}. We have bounded increments by the definition of the process (bounded range also works). So $\mathrm{E}[X_\tau] = E[X_0] = 0$ and the probability p of landing on $+a$ instead of $-b$ must satisfy $pa - (1-p)b = 0$, giving $p = \frac{b}{a+b}$.

Now suppose we want to find $\mathrm{E}[\tau]$. Let $Y_t = X_t^2 - t$. Then $Y_{t+1} = (X_t \pm 1)^2 - (t+1) = X_t^2 \pm 2X_t + 1 - (t+1) = (X_t^2 - t) \pm 2X_t = Y_t \pm 2X_t$. Since the plus and minus cases are equally likely, they cancel out in expectation and $\mathrm{E}[Y_{t+1}]|\mathcal{F}_t] = Y_t$: we just showed Y_t is a martingale.[2] We also show it has bounded increments (at least up until time τ), because $|Y_{t+1} - Y_t| = 2\,|X_t| \leq \max(a, b)$.

Using Corollary 6.3.3, $\mathrm{E}[Y_\tau] = 0$, which gives $\mathrm{E}[\tau] = \mathrm{E}[X_\tau^2]$. But we can calculate $\mathrm{E}[X_\tau^2]$: it is $a^2 \Pr[X_\tau = a] + b^2 \Pr[X_t = -b] = a^2(b/(a+b)) + b^2(a/(a+b)) = (a^2 b + b^2 a)/(a+b) = ab$.

[1] This is called a **random walk with two absorbing barriers**.

[2] This construction generalizes in a nice way to arbitrary martingales. Suppose $\{X_t\}$ is a martingale with respect to $\{\mathcal{F}_t\}$. Let $\Delta_t = X_t - X_{t-1}$, and let $V_t = \mathrm{Var}[\Delta_t|\mathcal{F}_{t-1}]$ be the conditional variance of the t-th increment (note that this is a random variable that may depend on previous outcomes). We can easily show that $Y_t = X_t^2 - \sum_{i=1}^{t} V_i$ is a

If we have a random walk that only stops at $+a$,[3] then if τ is the first time at which $X_\tau = a$, τ is a stopping time. However, in this case $\mathrm{E}[X_\tau] = a \neq \mathrm{E}[X_0] = 0$. So the optional stopping theorem doesn't apply in this case. But we have bounded increments, so Corollary 6.3.3 would apply if $\mathrm{E}[\tau] < \infty$. It follows that the expected time until we reach a is unbounded, either because sometimes we never reach a, or because we always reach a but sometimes it takes a very long time. [4]

We can also consider a **biased random walk** where $+1$ occurs with probability $p > 1/2$ and -1 with probability $q = 1 - p$. If X_t is the position of the random walk at time t, it isn't a martingale. But $Y_t = X_t - (p - q)t$ is, and it even has bounded increments. So if τ is the time at which $X_t = a$

martingale. The proof is that

$$\begin{aligned}
\mathrm{E}[Y_t|\mathcal{F}_{t-1}] &= \mathrm{E}[X_t^2 - \sum_{i=1}^{t} V_t|\mathcal{F}_{t-1}] \\
&= \mathrm{E}[(X_{t-1} + \Delta_t)^2|\mathcal{F}_{t-1}] - \sum_{i=1}^{t} V_i \\
&= \mathrm{E}[X_{t-1}^2 + 2X_{t-1}\Delta_t + \Delta_t^2|\mathcal{F}_{t-1}] - \sum_{i=1}^{t} V_i \\
&= X_{t-1}^2 + 2X_{t-1}\,\mathrm{E}[\Delta_t|\mathcal{F}_{t-1}] + \mathrm{E}[\Delta_t^2|\mathcal{F}_{t-1}] - \sum_{i=1}^{t} V_i \\
&= X_{t-1}^2 + 0 + V_t - \sum_{i=1}^{t} V_i \\
&= X_{t-1}^2 - \sum_{i=1}^{t-1} V_i \\
&= Y_{t-1}.
\end{aligned}$$

For the ± 1 random walk case, we have $V_t = 1$ always, giving $\sum_{i=1}^{t} V_i = t$ and $\mathrm{E}[X_\tau^2] = \mathrm{E}[X_0^2] + \mathrm{E}[\tau]$ when τ is a stopping time satisfying the conditions of the Optional Stopping Theorem. For the general case, the same argument gives $\mathrm{E}[X_\tau^2] = \mathrm{E}[X_0^2] + \mathrm{E}\left[\sum_{t=1}^{\tau} V_t\right]$ instead: the expected square position of X_t is incremented by the conditional variance at each step.

[3] This would be a **random walk with one absorbing barrier**.

[4] In fact, we always reach a. An easy way to see this is to imagine a sequence of intervals of length $n_1, n_2, \ldots$, where $n_{i+1} = \left(a + \sum_{j=1}^{i} n_j\right)^2$. At the end of the i-th interval, we are no lower than $-\sum_{j=0}^{i} n_j$, so we only need to go up $\sqrt{n_{i+1}}$ positions to reach a by the end of the $(i+1)$-th interval. Since this is just one standard deviation, it occurs with constant probability, so after a finite expected number of intervals, we will reach $+a$. Since there are infinitely many intervals, we reach $+a$ with probability 1.

and $\mathrm{E}[\tau]$ is finite,[5] the optional stopping theorem gives $\mathrm{E}[Y_\tau] = E[Y_0] = 0$, which gives $\mathrm{E}[a-(p-q)t] = 0$ or $\mathrm{E}[t] = \frac{a}{p-q}$, pretty much what we'd expect.

6.4.2 Wald's equation

Suppose we run a Las Vegas algorithm until it succeeds, and the i-th attempt costs X_i, where all the X_i are independent and identically distributed, and satisfy $0 \le X_i \le c$ for some c; note this implies $\mathrm{E}[X_i] = \mu$ for some μ and all i. Let τ be the number of times we run the algorithm; since we can tell when we are done, τ is a stopping time with respect to some filtration $\{\mathcal{F}_i\}$ to which the X_i are adapted.[6] Suppose also that $\mathrm{E}[\tau]$ exists. What is $\mathrm{E}\left[\sum_{i=1}^{\tau} X_i\right]$?

If τ were not a stopping time, this might be a very messy problem indeed. But when τ is a stopping time, we can apply it to the martingale $Y_t = \sum_{i=1}^{t}(X_i - \mu)$. This has bounded increments ($0 \le X_i \le c$, so $-c \le X_i - \mathrm{E}[X_i] \le c$), and we've already said $\mathrm{E}[\tau]$ is finite, so Corollary 6.3.3 applies. We thus have

$$\begin{aligned} 0 &= \mathrm{E}[Y_\tau] \\ &= \mathrm{E}\left[\sum_{i=1}^{\tau}(X_i - \mu)\right] \\ &= \mathrm{E}\left[\sum_{i=1}^{\tau} X_i\right] - \mathrm{E}\left[\sum_{i=1}^{\tau} \mu\right] \\ &= \mathrm{E}\left[\sum_{i=1}^{\tau} X_i\right] - \mathrm{E}[\tau]\mu. \end{aligned}$$

Rearranging this gives **Wald's equation**:

$$\mathrm{E}\left[\sum_{i=1}^{\tau} X_i\right] = \mathrm{E}[\tau]\mu. \tag{6.4.1}$$

6.4.3 Waiting times for patterns

Let's suppose we flip coins until we see some pattern appear; for example, we might flip coins until we see `HTHH`. What is the expected number of coin-flips until this happens?

[5] Exercise: Show $\mathrm{E}[\tau]$ is finite.

[6] A stochastic process $\{X_t\}$ is **adapted** to a filtration $\{\mathcal{F}_t\}$ if each X_t is measurable $\mathcal{F}_t$.

A very clever trick due to Li [Li80] solves this problem exactly using the Optional Stopping Theorem. Suppose our pattern is $x_1 x_2 \dots x_k$. We imagine an army of gamblers, one of which shows up before each coin-flip. Each gambler starts by borrowing \$1 and betting it that the next coin-flip will be x_1. If she wins, she takes her \$2 and bets that the next coin-flip will be x_2, continuing to play double-or-nothing until either she loses (and is out her initial \$1) or wins her last bet on x_k (and is up $2^k - 1$). Because each gambler's winnings form a martingale, so does their sum, and so the expected total return of all gamblers up to the stopping time τ at which our pattern first occurs is 0.

We can now use this fact to compute $\mathrm{E}[\tau]$. When we stop at time τ, we have one gambler who has won $2^k - 1$. We may also have other gamblers who are still in play. For each i with $x_1 \dots x_i = x_{k-i+1} \dots x_k$, there will be a gambler with net winnings $2^i - 1$. The remaining gamblers will all be at -1.

Let $\chi_i = 1$ if $x_1 \dots x_i = x_{k-i+1} \dots x_k$, and 0 otherwise. Then

$$\begin{aligned}
\mathrm{E}[X_\tau] &= \mathrm{E}\left[-\tau + \sum_{i=1}^{k} \chi_i 2^i\right] \\
&= -\mathrm{E}[\tau] + \sum_{i=1}^{k} \chi_i 2^i \\
&= 0.
\end{aligned}$$

It follows that $\mathrm{E}[\tau] = \sum_{i=1}^{k} \chi_i 2^i$.

For example, the pattern `HTHH` only overlaps with its prefix `H` so in this case we have $\mathrm{E}[\tau] = \sum \chi_i 2^i = 2 + 16 = 18$. But `HHHH` overlaps with all of its prefixes, giving $\mathrm{E}[\tau] = 31$ in this case. At the other extreme, `THHH` has no overlap and gives $\mathrm{E}[\tau] = 16$.

This analysis generalizes in the obvious way to biased coins and larger alphabets; see the paper [Li80] for details.

Chapter 7

Markov chains

A **stochastic process** is a sequence of random variables $\{X_t\}$, where we think of X_t as the value of the process at time t.

There are two stochastic processes that come up over and over again in the analysis of randomized algorithms. One is a martingale, where the next increment may depend in a complicated way on the past history but has expectation 0; the other is a Markov chain, where the next step depends only on the current location and not the previous history. In this chapter, we'll give basic definitions for Markov chains, and then talk about the most useful algorithmic property, which is convergence to a fixed distribution on states after sufficiently many steps in many Markov chains.

7.1 Basic definitions and properties

A **Markov chain** or **Markov process** is a stochastic process where the distribution of X_{t+1} depends only on the value of X_t and not any previous history. (Formally, for all sets of states A, $\mathrm{E}[[X_{t+1} \in A]|X_0 \ldots X_t] = \mathrm{E}[[X_{t+1} \in A]|X_t]$; contrast with the martingale property.) The **state space** of the chain is just the set of all values that each X_t can have. A Markov chain is **finite** or **countable** if it has a finite or countable state space, respectively. We'll mostly be interested in countable Markov chains. We'll also assume that our Markov chains are **homogeneous**, which means that $\Pr[X_{t+1} = j|X_t = i]$ doesn't depend on t.

For a countable Markov chain, we can describe its behavior completely by giving the state space and the one-step **transition probabilities** $p_{ij} = \Pr[X_{t+1} = j|X_t = i]$. Given p_{ij}, we can calculate two-step transition probabilities $p_{ij}^{(2)} = Pr[X_{t+2} = j|X_t = i] = \sum_k Pr[X_{t+2} = j|X_{t+1} = k]Pr[X_{t+1} =$

$k|X_t = i] = \sum_k p_{ik}p_{kj}$. This is identical to the formula for matrix multiplication, and so we can think of the transition probabilities as given by a transition matrix P, then $p_{ij}^{(2)} = (P^2)_{ij}$ and in general the n-step transition probability $p_{ij}^{(n)} = (P^n)_{ij}$.

Conversely, given any matrix with non-negative entries where the rows sum to 1 ($\sum_j P_{ij} = 1$, or $P1 = 1$), there is a corresponding Markov chain given by $p_{ij} = P_{ij}$. Such a matrix is called a **stochastic matrix**.

The general formula for $(n + m)$-step transition probabilities is that $p_{ij}^{(n+m)} = \sum_k p_{ik}^{(n)} p_{kj}^{(m)}$. This is known as the **Chapman-Kolmogorov equation** and is equivalent to the matrix identity $P^{n+m} = P^n P^m$.

A distribution over states of the Markov chain at some time t can be given by a row vector x, where $x_i = Pr[X_t = i]$. To compute the distribution at time $t + 1$, we use the law of total probability: $\Pr[X_{t+1} = j] = \sum_i Pr[X_t = i]Pr[X_{t+1} = j|X_t = i] = \sum_i x_i p_{ij}$. Again we have the formula for matrix multiplication (where we treat x as a $1 \times i$ matrix); so the distribution vector at time $t + 1$ is just xP, and at time $t + n$ is xP^n.

We like Markov chains for two reasons:

1. They describe what happens in a randomized algorithm; the state space is just the set of all states of the algorithm, and the Markov property holds because the algorithm can't remember anything that isn't part of its state. So if we want to analyze randomized algorithms, we will need to get good at analyzing Markov chains.

2. They can be used to do sampling over interesting distributions. Under appropriate conditions (see below), the state of a Markov chain converges to a **.** If we build the right Markov chain, we can control what this stationary distribution looks like, run the chain for a while, and get a sample close to the stationary distribution.

In both cases we want to have a bound on how long it takes the Markov chain to converge, either because it tells us when our algorithm terminates, or because it tells us how long to mix it up before looking at the current state.

7.1.1 Examples

- A fair ± 1 random walk. The state space is $\mathbb{Z}$, the transition probabilities are $p_{ij} = 1/2$ if $|i - j| = 1$, 0 otherwise. This is an example of a Markov chain that is also a martingale.

- A fair ± 1 random walk on a cycle. As above, but now the state space is $\mathbb{Z}/m$, the integers mod m. An example of a finite Markov chain.
- Random walks with absorbing/reflecting barriers.
- Random walk on a graph $G = (V, E)$. The state space is V, the transition probalities are $p_{uv} = 1/d(u)$ if $uv \in E$. (One can also have more general transition probabilities.)
- The Markov chain given by $X_{t+1} = X_t + 1$ with probability $1/2$, and 0 with probability $1/2$. The state space is $\mathbb{N}$.
- 2-SAT algorithm. State is a truth-assignment. The transitional probabilities are messy but arise from the following process: pick an unsatisfied clause, pick one of its two variables uniformly at random, and invert it. Then there is an absorbing state at any satisfying assignment. (A similar process works for 2-colorability, 3-SAT, 3-colorability, etc.; note that for the **NP**-hard problems, it may take a while to reach an absorbing state. The constructive Lovász Local Lemma proof from Section 4.3.5 also follows this pattern.)

7.1.2 Classification of states

Given a Markov chain, define $f_{ij}(n) = \Pr[j \notin \{X_1 \dots X_{n-1}\} \wedge j = X_n | X_0 = i]$; $f_{ij}(n)$ is the probability that the **first passage time** from i to j equals n. Define $f_{ij} = \sum_{n=1}^{\infty} f_{ij}(n)$; this is the probability that the chain ever reaches j starting from i. A state is **persistent** if $f_{ii} = 1$ and **transient** if $f_{ii} < 1$.

Finally, define the **mean recurrence time** $\mu_i = \sum_{n=1}^{\infty} n f_{ii}(n)$ if i is persistent and $\mu_i = \infty$ if i is transient. We use the mean recurrence time to further classify persistent states. if the sum diverges, we say that the state is **null persistent**; otherwise it is **non-null persistent**.

These definitions are summarized in Tables 7.1 and 7.2.

Examples: In a random walk with absorbing barriers at $\pm n$, states $-n$ and $+n$ are (non-null) persistent, and the rest of the states are transient. In general, any state in a finite chain is either non-null persistent or transient, and there is at least one persistent state. In a random walk on $\mathbb{Z}$, all states are null persistent. In the process on $\mathbb{N}$ where $X_{t+1} = X_t + 1$ with probability 1, all states are transient.

The **period** $d(i)$ of a state i is $\gcd(\{n : p_i(n) > 0\})$. A state i is **aperiodic** if $d(i) = 1$, and **periodic** otherwise. A Markov chain is aperiodic if all its states are aperiodic.

Parameter	Name	Definition
$p_{ij}(n)$	n-step transition probability	$\Pr[X_{t+n} = j \mid X_t = i]$
p_{ij}	Transition probability	$\Pr[X_{t+1} = j \mid X_t = i] = p_{ij}(1)$
$f_{ij}(n)$	First passage time	$\Pr[j \notin \{X_1 \dots X_n - 1\} \wedge X_n = j \mid X_0 = i]$
f_{ij}	Probability of reaching j from i	$\sum_{n=1}^{\infty} f_{ij}(n)$
μ_i	Mean recurrence time	$\mu_i = \sum_{n=1}^{\infty} n f_{ii}(n)$
π_i	Stationary distribution	$\pi_j = \sum_i \pi_i p_{ij} = 1/\mu_j$

Table 7.1: Markov chain parameters

f_{ii}	μ_i	classification
< 1	$= \infty$	transient
$= 1$	$= \infty$	null persistent
$= 1$	$< \infty$	non-null persistent

Table 7.2: Classification of Markov chain states

The most well-behaved states are the aperiodic non-null persistent states; these are called **ergodic**. A Markov chain is ergodic if all its states are.

7.1.3 Reachability

State i **communicates** with state j if $p_{ij}(n) > 0$ for some n; i.e., it is possible to reach j from i. This is often abbreviated as $i \to j$. Two states i and j **intercommunicate** if $i \to j$ and $j \to i$; this is similarly abbreviated as $i \leftrightarrow j$. If $i \leftrightarrow j$, it's not hard to show that i and j have the same period and classification. A set of states S is **closed** if $p_{ij} = 0$ for all $i \in S$, $j \notin S$, and **irreducible** if $i \leftrightarrow j$ for all i, j in S.

Using graph-theoretic terminology, $i \to j$ if j is **reachable** from i (through edges corresponding to transitions with nonzero probabilities), and a set of states is closed if it is **strongly connected**. We can thus decompose a Markov chain into strongly-connected components, and observe that all states in a particular strongly-connected component are persistent if and only if the strongly-connected component has no outgoing edges (i.e., is closed). For finite chains there must be at least one closed strongly-connected component (corresponding to a sink in the quotient DAG); this gives an instant proof that finite chains have persistent states.

If the entire chain forms a single strongly-connected component, we say it is **irreducible**.

7.2 Stationary distributions

The key useful fact about irreducible Markov chains is that (at least in the finite case) they have **stationary distributions**, where a stationary distribution is a distribution π (non-negative row vector summing to 1) such that $\pi P = \pi$. The full theorem is:

Theorem 7.2.1. *An irreducible Markov chain has a stationary distribution π if and only if all states are non-null persistent, and if π exists, then $\pi_i = 1/\mu_i$ for all i.*

The proof of this theorem takes a while, so we won't do it (see [GS01, §6.4] if you want to see the details). For finite chains, the first part follows from the **Perron-Frobenius theorem**,[1] which says that stochastic matrices always have an eigenvector corresponding to an eigenvalue of 1. The second part ($\pi_i = 1/\mu_i$) is just a special case of the **renewal theorem**.[2]

For non-irreducible Markov chains, there is a stationary distribution on each closed irreducible subset, and the stationary distributions for the chain as a whole are all convex combinations of these stationary distributions.

Examples: In the random walk on $\mathbb{Z}_m$ the stationary distribution satisfies $\pi_i = 1/m$ for all i (immediate from symmetry). By contrast, the random walk on $\mathbb{Z}$ has no stationary distribution (the states are all null persistent). The process on $\mathbb{N}$ where $p_{ij} = 1/2$ for $j = i + 1$ or $j = 0$ has a stationary distribution π given by $\pi_i = 2^{-i-1}$; this is an example of an infinite chain that nonetheless has only non-null persistent states. For a random walk with absorbing barriers at $\pm n$, there is no unique stationary distribution; instead, any vector π where $\pi_i = 0$ unless $i = \pm n$ and $\pi 1 = 1$ is stationary.

Note that it's generally not a good strategy to compute π by computing μ first; instead, solve the matrix equation $\pi P = \pi$ by rewriting it as $\pi(P - I) = 0$ and adding the constraint $\pi 1 = 1$, and then compute $\mu_i = 1/p_i$.

7.2.1 The ergodic theorem

The basic version of the **ergodic theorem** says that if an aperiodic Markov chain has a stationary distribution π (i.e., it's ergodic), then it converges to π if we run it long enough. This is not terribly hard to prove, using a very clever technique called **coupling**, where we take two copies of the chain, once of which starts in an arbitrary distribution, and one of which starts in the stationary distribution, and show that we can force them to converge

[1] `http://en.wikipedia.org/wiki/Perron-Frobenius_theorem`.

[2] `http://en.wikipedia.org/wiki/Renewal_theory`.

to each other by carefully correlating their transitions. Since coupling is a generally useful technique for bounding rate of convergence, we'll give the proof of the ergodic theorem below.

A more general version of the ergodic theorem says that for any Markov chain and any aperiodic state j, then $p_{jj}(n) \to 1/\mu_j$ and $p_{ij}(n) \to f_{ij}/\mu_j$. This allows us in principle to compute the limit distribution for any aperiodic Markov chain with a given starting distribution x by summing $\sum_i x_i f_{ij}/\mu_j$. It also implies that transient and non-null persistent states vanish in the limit.

7.2.1.1 Proof

We'll do this by constructing a **coupling** between two copies of the Markov chain, which as stated before is a joint distribution on two processes that tends to bring them together while looking like the original process when restricted to one sub-process or the other. In the first process $\{X_t\}$, we start with an arbitrary initial distribution for X_0. In the second $\{Y_t\}$, we start with $\Pr[Y_t = i] = \pi_i$ for all i. We now evolve the joint process $\{(X_t, Y_t)\}$ by the rule $(i, i') \to (j, j')$ with probability $p_{ij}p_{i'j'}$ if $i \neq i'$ and $(i, i') \to (j, j')$ with probability p_{ij} if $i = i'$ and $j = j'$ and 0 if $i = i'$ and $j \neq j'$. In other words, we let X_t and Y_t both wander around independently until they collide, after which they stick together and wander around together.

The reason this shows convergence is that we can write

$$\Pr[X_t = j] = \Pr[X_t = Y_t] \Pr[Y_t = j | X_t = Y_t] + \Pr[X_t \neq Y_t] \Pr[X_t = j | X_t \neq Y_t]$$

and similarly

$$\Pr[Y_t = j] = \Pr[X_t = Y_t] \Pr[Y_t = j | X_t = Y_t] + \Pr[X_t \neq Y_t] \Pr[Y_t = j | X_t \neq Y_t].$$

The sneaky bit here is that in the case where $X_t = Y_t$, we can write both events in terms of $\Pr[Y_t = j | X_t = Y_t]$.

Suppose now that $\Pr[X_t = Y_t] \to 1$. Then the first equation gives $\Pr[X_t = j] - \Pr[Y_t = j | X_t = Y_t] \to 0$ and the second gives $\Pr[Y_t = j] - \Pr[Y_t = j | X_t = Y_t] \to 0$. Since both $\Pr[X_t = j]$ and $\Pr[Y_t = j]$ converge to the same value, they converge to each other, giving $\Pr[X_t = j] \to \Pr[Y_t = j] = \pi_j$.

So now we just need to show $\Pr[X_t = Y_t] \to 1$. Without loss of generality, suppose that $X_0 = j$, where j is a non-null persistent aperiodic state. With probability π_j, $X_0 = Y_0$ and we are done. If not, let $i = Y_0$ and pick some future time n for which $p_{ij}^{(n)} > 0$. If we are very lucky, $p_j^{(n)}$ is also greater

than 0, and so we get a nonzero probability of collision. If we are not so lucky, let m be some value such that $p_j^{(n+m)} > 0$ and $p_j^{(m)} > 0$ (if no such value exists, then j is periodic). Then we have

$$\begin{aligned}\Pr[X_t = Y_t = j] &\geq \Pr[X_t = j]\Pr[Y_t = j] \\ &= p_{ij}^{(n)} p_{jj}^{(m)} p_{jj}^{(n+m)} \\ &> 0.\end{aligned}$$

So there is a finite time $n + m$ after which we get some nonzero probability that X_t and Y_t collide. If we do not collide, repeat the argument starting with $i = Y_{n+m}$. If we let k be the maximum value of $n + m$ taken over all initial states i, and $p > 0$ be the minimum probability of collision over all initial states i after k steps, then we have that $\Pr[X_{\alpha k} \neq Y_{\alpha k}] \leq (1-p)^\alpha \to 0$.

7.2.2 Reversible chains

Some chains have the property of being **reversible**. Formally, a chain with transition probabilities p_{ij} is reversible if there is a distribution π such that

$$\pi_i p_{ij} = \pi_j p_{ji} \tag{7.2.1}$$

for all i, j. These are called the **detailed balance equations**—they say that in the stationary distribution, the probability of seeing a transition from i to j is equal to the probability of seeing a transition from j to i). If this is the case, then $\sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j$, which means that π is stationary.

This often gives a very quick way to compute the stationary distribution, since if we know π_i, and $p_{ij} \neq 0$, then $\pi_j = \pi_i p_{ij}/p_{ji}$. If the transition probabilities are reasonably well-behaved (for example, if $p_{ij} = p_{ij}$ for all i, j), we may even be able to characterize the stationary distribution up to a constant multiple even if we have no way to efficiently enumerate all the states of the process.

The reason that such a chain is called reversible is that if we start in the stationary distribution at time 0, then the sequence of random variables $(X_0, \ldots, X_t)$ has *exactly the same distribution* as the reversed sequence $(X_t, \ldots, X_0)$. [3]

Note that a reversible chain can't have a period higher than 2, since we can always step back to where we came from.

[3] Proof: Start with $\Pr[\forall i X_i = x_i] = \pi_{x_0} \prod_{i=0}^{t} p_{x_i x_{i+1}}$. Now observe that we can move the π_{x_0} across the first factor to get $p_{x_1 x_0} \pi_{x_1} \prod_{i=1}^{t} p_{x_i x_{i+1}}$ and in general $\left(\prod_{i=0}^{j-1} p_{x_{i+1} x_i}\right) \pi_{x_j} \left(\prod_{i=j}^{t} p_{x_i x_{i+1}}\right)$. At $j = t$ we get $\pi_{x_t} \prod_{i=0}^{t-1} p_{x_{i+1} x_i} = \Pr[\forall i X_i = x_{t-i}]$.

7.2.2.1 Basic examples

Random walk on a graph Given two adjacent vertices u and v, we have $\pi_v = \pi_u d(v)/d(u)$. This suggests $\pi_v = cd(v)$ for some constant c and all v (the stationary distribution is proportional to the degree), and the constant c is determined by $\sum_v \pi_v = c\sum d(v) = 1$, giving $\pi_v = d(v)/\sum_u d(u)$.

Random walk on a weighted graph Here each edge has a weight w_{uv} where $0 < w_{uv} = w_{vu} < \infty$, with self-loops permitted. A step of the random walk goes from u to v with probability $w_{uv}/\sum_{v'} w_{uv'}$. It is easy to show that this random walk has stationary distribution $\pi_u = \sum_u w_{uv}/\sum_u \sum_v w_{uv}$, generalizing the previous case, and that the resulting Markov chain satisfies the detailed balance equations.

Random walk with uniform stationary distribution Now let d be the maximum degree of the graph, and traverse each each with probability $1/d$, staying put on each vertex u with probability $1 - d(u)/d$. The stationary distribution is uniform, since for each pair of vertices u and v we have $p_{uv} = p_{vu} = 1/d$ if u and v are adjacent and 0 otherwise.

7.2.2.2 Metropolis-Hastings

The basic idea of the **Metropolis-Hastings algorithm**[MRR+53, Has70] (sometimes just called **Metropolis**) is that we start with a reversible Markov chain P with a known stationary distribution π, but we'd rather get a chain Q on the same states with a different stationary distribution μ, where $\mu_i = f(i)/\sum_j f(j)$ is proportional to some function $f \geq 0$ on states that we can compute easily.

A typical application is that we want to sample according to $\Pr[i|A]$, but A is highly improbable (so we can't just use **rejection sampling**, where we sample random points from the original distribution until we find one for which A holds), and $\Pr[i|A]$ is easy to compute for any fixed i but tricky to compute for arbitrary events (so we can't use divide-and-conquer). If we let $f(i) \propto \Pr[i|A]$, then Metropolis-Hastings will do exactly what we want, assuming it converges in a reasonable amount of time.

Let q be the transition probability for Q. Define, for $i \neq j$,

$$\begin{aligned} q_{ij} &= p_{ij} \min\left(1, \frac{\pi_i f(j)}{\pi_j f(i)}\right) \\ &= p_{ij} \min\left(1, \frac{\pi_i \mu_j}{\pi_j \mu_i}\right) \end{aligned}$$

and let q_{ii} be whatever probability is left over. Now consider two states i and j, and suppose that $\pi_i f(j) \geq \pi_j f(i)$. Then

$$q_{ij} = p_{ij}$$

which gives

$$\mu_i q_{ij} = \mu_i p_{ij},$$

while

$$\begin{aligned}
\mu_j q_{ji} &= \mu_j p_{ji}(\pi_j \mu_i / \pi_i \mu_j) \\
&= p_{ji}(\pi_j \mu_i / \pi_i) \\
&= \mu_i (p_{ji} \pi_j) / \pi_i \\
&= \mu_i (p_{ij} \pi_i) / \pi_i \\
&= \mu_i p_{ij}
\end{aligned}$$

(note the use of reversibility of P in the second-to-last step). So we have $\mu_j q_{ji} = \mu_i p_{ij} = \mu_i q_{ij}$ and Q is a reversible Markov chain with stationary distribution μ.

We can simplify this when our underlying chain P has a uniform stationary distribution (for example, when it's the random walk on a graph with maximum degree d, where we traverse each edge with probability $1/d$). Then we have $\pi_i = \pi_j$ for all i, j, so the new transition probabilities q_{ij} are just $\frac{1}{d} \min(1, f(j)/f(i))$. Most of our examples of reversible chains will be instances of this case (see also [MU05, §10.4.1]).

7.3 Bounding convergence rates using the coupling method

In order to use the stationary distribution of a Markov chain to do sampling, we need to have a bound on the rate of convergence to tell us when it is safe to take a sample. There are two standard techniques for doing this: coupling, where we show that a copy of the process starting in an arbitrary state can be made to converge to a copy starting in the stationary distribution; and spectral methods, where we bound the rate of convergence by looking at the second-largest eigenvalue of the transition matrix. We'll start with coupling because it requires less development.

(See also [Gur00] for a survey of the relationship between the various methods.)

Note: these notes will be somewhat sketchy. If you want to read more about coupling, a good place to start might be Chapter 11 of [MU05]; Chapter 4-3 (`http://www.stat.berkeley.edu/~aldous/RWG/Chap4-3.pdf`) of the unpublished but nonetheless famous **Aldous-Fill manuscript** (`http://www.stat.berkeley.edu/~aldous/RWG/book.html`), which is a good place to learn about Markov chains and Markov chain Monte Carlo methods in general; or even an entire book [Lin92]. We'll mostly be using examples from the Aldous-Fill text.

7.3.1 The basic coupling lemma

The same sort of coupling trick that proves eventual convergence to π can in some cases be used to prove fast convergence. The idea is that we can bound the **total variation distance** $d_{TV}(p,\pi) = \max_A |\Pr_p[A] - \Pr_\pi[A]| = \sum_i \max(p_i - \pi_i, 0) = \frac{1}{2}\sum_i |p_i - \pi_i|$ by bounding $\Pr[X_t \neq Y_t]$, where X_t is a copy of the process that starts in some arbitrary distribution and Y_t is a copy that starts in π. As before, we start with the fact that, for all j,

$$\Pr[X_t = j] = \Pr[X_t = Y_t]\Pr[Y_t = j | X_t = Y_t] + \Pr[X_t \neq Y_t]\Pr[X_t = j | X_t \neq Y_t]$$

and

$$\Pr[Y_t = j] = \Pr[X_t = Y_t]\Pr[Y_t = j | X_t = Y_t] + \Pr[X_t \neq Y_t]\Pr[Y_t = j | X_t \neq Y_t].$$

Again we use the sneaky fact that when $X_t = Y_t$, we can write both events in terms of $\Pr[Y_t = j | X_t = Y_t]$.

Subtract to get

$$\Pr[X_t = j] - \Pr[Y_t = j] = \Pr[X_t \neq Y_t](\Pr[X_t = j | X_t \neq Y_t] - \Pr[Y_t = j | X_t \neq Y_t]). \tag{7.3.1}$$

Now observe that the parenthesized term is somewhere in the range -1 to 1, giving, for each j,

$$|Pr[X_t = j] - Pr[Y_t = j]| \leq \Pr[X_t \neq Y_t].$$

But in fact we can do much better than this. If we sum (7.3.1) over all j, we get

$$\begin{aligned}\sum_j |\Pr[X_t = j] - \Pr[Y_t = j]| &= \Pr[X_t \neq Y_t] \sum_j |\Pr[X_t = j | X_t \neq Y_t] - \Pr[Y_t = j | X_t \neq Y_t]| \\ &\leq \Pr[X_t \neq Y_t] \sum_j (\Pr[X_t = j | X_t \neq Y_t] + \Pr[Y_t = j | X_t \neq Y_t]) \\ &\leq 2 \Pr[X_t \neq Y_t].\end{aligned}$$

So the total variation distance between the distributions of X_t and Y_t is bounded by half of this, or $\Pr[X_t \neq Y_t]$.

This argument can be a bit confusing, so it may help to think about it in reverse: We can write the event $[X_t \neq Y_t]$ as the union of disjoint events $[X_t = i \wedge Y_t \neq i]$. If $\Pr[X_t = i] > \Pr[Y_t = i]$, then $\Pr[X_t = i] - \Pr[Y_t = i]$ gives a lower bound on $\Pr[X_t = i \wedge Y_t \neq i]$, no matter how thoroughly X_t and Y_t are correlated. So $\sum_i \max(0, \Pr[X_t = i] - \Pr[Y_t = i]$ (one of the versions of total variation distance) is a lower bound on $\Pr[X_t \neq Y_t]$. But this means $\Pr[X_t \neq Y_t]$ is an upper bound on total variation distance, no matter what we did to try to correlate X_t and Y_t.

However we argue this bound, we can now go hunting for couplings that make $X_t = Y_t$ with high probability for large enough t. [4]

7.3.2 Random walk on a cycle

Let's suppose we do a random walk on $\mathbb{Z}_m$, where to avoid periodicity at each step we stay put with probability $1/2$, move counterclockwise with probability $1/4$, and move clockwise with probability $1/4$. To make our lives slightly easier, we'll assume m is even. What's a good choice for a coupling to show this process converges quickly?

We'll assume we choose X's move first and then try to figure out Y's move in response. If $X_t = Y_t$, we'll just have Y move with X (the usual outcome). If $X_t - Y_t$ is even, we'll have $Y_{t+1} - Y_t = -(X_{t+1} - X_t)$; X and

[4] A curious fact: in principle there *always* exists a coupling between X and Y such that $d_{TV}(X_t, Y_t) = \Pr[X_t \neq Y_t]$. The intuition is that we can take trajectories in the X and Y processes that end in the same state and match them up as much as we can; done right, this makes the event $X_t = Y_t$ occur as often as possible given the mismatches between $\Pr[X_t = x]$ and $\Pr[Y_t = x]$. The only problem with this is that (a) we may have to know a lot about how the convergence comes about to know which trajectories to follow, because we are essentially picking the entire path for both processes at time 0; and (b) we may have to correlate the initial states X_0 and Y_0, which is easy if X_0 puts all its weight on one point, but gets complicated if the distribution on X_0 is not trivial. So this fact is not particularly useful in practice.

Y move in opposite directions. If $X_t - Y_t$ is odd, we'll have Y move if and only if X doesn't; this makes the distance between them (in both directions, because m is even) even after one step, and the other transitions will keep the distance between them even. In each case the movement of Y follows the distribution given by the underlying Markov chain; we just happen to correlate it in a somewhat convoluted way with the movement of X.

Now let us ask how long it takes on average until X_t and Y_t collide. We have at most one step to make the distance between them even. After that point, the distance increases or decreases by exactly 2 with probability 1/4 each, and stays the same with probability 1/2. If we consider only the steps where the distance changes, we have a random ± 2 walk with absorbing barriers at 0 and m, which is equivalent to a random ± 1 walk with absorbing barriers at 0 and $m/2$; the worst-case starting position for this second walk is exactly in the middle, giving an expected time to reach a barrier of $(m/4)^2 = m^2/16$. But we have to multiply this by 2 to account for the fact that we only take a step every other time unit on average, and add one for the possible initial parity-fixing step, which gives a bound of $m^2/8 + 1$ on the expected time to collide. Using Markov's inequality, after $m^2/4 + 2$ steps we have $\Pr[X_t \neq Y_t] \leq 1/2$, and by iterating this argument, after $\alpha m^2/4$ steps we will have $\Pr[X_t \neq Y_t] \leq 2^{-\alpha}$. This gives a **mixing time** to reach $d_{TV} \leq \epsilon$ of at most $(m^2/4 + 2)\lg(1/\epsilon)$, which means that the process is **rapidly mixing**: the mixing time is polynomial in the size of the underlying process and $\log(1/\epsilon)$.[5]

It's worth noting that this example was very carefully rigged to make the coupling argument clean. It still works (perhaps with a slight change in the bound) if m is not even or $\Pr[X_{t+1} = X_t] \neq \Pr[X_{t+1} \neq X_t]$, but the details are messier.

7.3.3 Random walk on a hypercube

Start with a bit-vector of length n. At each step, choose an index uniformly at random, and set the value of the bit-vector at that index to 0 or 1 with equal probability. How long until we get a nearly-uniform distribution over all 2^n possible bit-vectors?

Here we apply the same transformation to both the X and Y vectors.

[5]The choice of 2 for the constant in Markov's inequality is probably not optimal. Suppose the expected **coupling time** at which the two processes are first equal is T. If we restart the process every cT steps, then at time t we have a total variation bounded by of $c^{-\lfloor t/cT \rfloor}$. The expression $c^{-t/cT}$ is minimized by minimizing $c^{-1/c}$ or equivalently $-lnc/c$, which occurs at $c = e$. This gives a time to reach ϵ of $Te\ln(1/\epsilon)$.

It's easy to see that the two vectors will be equal once every index has been selected once. The waiting time for this to occur is just the waiting time nH_n for the coupon collector problem. We can either use this expected time directly to show that the process mixes in time $O(n \log n \log(1/\epsilon))$ as above, or we can use known sharp concentration bounds on coupon collector (see for example [MR95, §3.6.3], which shows $\lim_{n\to\infty} \Pr[T \geq n(\ln n + c)] = 1 - \exp(-\exp(-c))$ to show that in the limit $n \ln n + n \ln \ln(1/(1-\epsilon)) = n \ln n + O(n \log(1/\epsilon))$ is enough.[6]

We can improve the bound slightly by observing that, on average, half the bits in X_0 and Y_0 are already equal; doing this right involves summing over a lot of cases, so we won't do it.

7.3.4 Various shuffling algorithms

Here we have a deck of n cards, and we repeatedly apply some random transformation to the deck to converge to a stationary distribution that is uniform over all permutations of the cards (usually this is obvious by symmetry, so we won't bother proving it). Our goal is to show that the expected **coupling time** at which our deck ends up in the same permutation as an initially-stationary deck is small. We do this by counting how many cards S_t are in the same position in both decks, and showing that, for a suitable coupling, (a) this quantity never decreases, and (b) it increases with some nonzero probability at each step. The expected coupling time is then $\sum_k 1/\Pr[S_{t+1} = k+1 | S_t = k]$.

Move-to-top This is a variant of card shuffling that is interesting mostly because it gives about the easiest possible coupling argument. At each step, we choose one of the cards uniformly at random (including the top card) and move it to the top of the deck. How long until the deck is fully shuffled, i.e. until the total variation distance between the actual distribution and the stationary distribution is bounded by ϵ?

Here the trick is that when we choose a card to move to the top in the X process, we choose the same card in the Y process. It's not hard to see that this links the two cards together so that they are always in the same position in the deck in all future states. So to keep track of how well the coupling is working, we just keep track of how many cards are linked in this way, and observe that as soon as $n-1$ are, the two decks are identical.

[6] This is a little tricky; we don't know from this bound alone how fast the probability converges as a function of n, so to do this right we need to look into the bound in more detail.

Note: Unlike some of the examples below, we don't consider two cards to be linked just because they are in the same position. We are only considering cards that have gone through the top position in the deck (which corresponds to some initial segment of the deck, viewed from above). The reason is that these cards never become unlinked: if we pick two cards from the initial segment, the cards above them move down together. But deeper cards that happen to match might become separated if we pull a card from one deck that is above the matched pair while its counterpart in the other deck is below the matched pair.

Having carefully processed the above note, given k linked cards the probability that the next step links another pair of cards is exactly $(n-k)/n$. So the expected time until we get $k+1$ cards is $n/(n-k)$, and if we sum these waiting times for $k = 0 \dots n-1$, we get nH_n, the waiting time for the coupon collector problem. So the bound on the mixing time is the same as for the random walk on a hypercube.

Random exchange of arbitrary cards Here we pick two cards uniformly and independently at random and swap them. (Note there is a $1/n$ chance they are the same card; if we exclude this case, the Markov chain has period 2.) To get a coupling, we reformulate this process as picking a random card and a random location, and swapping the chosen card with whatever is in the chosen location in both the X and Y processes.

First let's observe that the number of linked cards never decreases. Let x_i, y_i be the position of card i in each process, and suppose $x_i = y_i$. If neither card i nor position x_i is picked, i doesn't move, and so it stays linked. If card i is picked, then both copies are moved to the same location; it stays linked. If position x_i is picked, then it may be that i becomes unlinked; but this only happens if the card j that is picked has $x_j \neq y_j$. In this case j becomes linked, and the number of linked cards doesn't drop.

Now we need to know how likely it is that we go from k to $k+1$ linked cards. We've already seen a case where the number of linked cards increases; we pick two cards that aren't linked and a location that contains cards that aren't linked. The probability of doing this is $((n-k)/n)^2$, so our total expected waiting time is $n^2 \sum (n-k)^{-2} = n^2 \sum k^{-2} \leq n\pi^2/6$ (see `http://en.wikipedia.org/wiki/Basel_problem` for an extensive discussion of the useful but non-obvious fact that $\sum_{k \in \mathbb{N}_+} k^{-2} = \zeta(2) = \pi^2/6$.) The final bound is $O(n^2 \log(1/\epsilon))$.

This bound is much worse that the bound for move-to-top, which is surprising. In fact, the real bound is $O(n \log n)$ with high probability,

although the proof uses very different methods (see `http://www.stat.berkeley.edu/~aldous/RWG/Chap7.pdf`). This shows that the coupling method doesn't always give tight bounds (perhaps we need a better coupling?).

Random exchange of adjacent cards Suppose now that we only swap adjacent cards. Specifically, we choose one of the n positions i in the deck uniformly at random, and then swap the cards a positions i and $i+1 \pmod n$ with probability $1/2$. (The $1/2$ is there for the usual reason of avoiding periodicity.)

So now we want a coupling between the X and Y processes where each possible swap occurs with probability $\frac{1}{2n}$ on both sides, but somehow we correlate things so that like cards are pushed together but never pulled apart. The trick is that we will use the same position i on both sides, but be sneaky about when we swap. In particular, we will aim to arrange things so that once some card is in the same position in both decks, both copies move together, but otherwise one copy changes its position by ± 1 relative to the other with a fixed probability $\frac{1}{2n}$.[7]

The coupled process works like this. Let D be the set of indices i where the same card appears in both decks at position i or at position $i+1$. Then we do:

1. For $i \in D$, swap $(i, i+1)$ in both decks with probability $\frac{1}{2n}$.
2. For $i \notin D$, swap $(i, i+1)$ in the X deck only with probability $\frac{1}{2n}$.
3. For $i \notin D$, swap $(i, i+1)$ in the Y deck only with probability $\frac{1}{2n}$.
4. Do nothing with probability $\frac{|D|}{2n}$.

It's worth checking that the total probability of all these events is $|D|/2n + 2(n-|D|)/2n + |D|/2n = 1$. More important is that if we consider only one of the decks, the probability of doing a swap at $(i, i+1)$ is exactly $\frac{1}{2n}$ (since we catch either case 1 or 2 for the X deck or 1 or 3 for the Y deck).

Now suppose that some card c is at position x in X and y in Y. If $x = y$, then both x and $x-1$ are in D, so the only way the card can move is if it moves in both decks: linked cards stay linked. If $x \neq y$, then c moves in deck X or deck Y, but not both. (The only way it can move in both is in case 1, where $i = x$ and $i+1 = y$ or vice verse; but in this case i can't

[7] This last step is what I bungled in class on 2011-03-01.

be in D since the copy of c at position x doesn't match whatever is in deck Y, and the copy at position y doesn't match what's in deck X.) In this case the distance $x - y$ goes up or down by 1 with equal probability $1/2n$. Considering $x - y \pmod n$, we have a "lazy" random walk that moves with probability $1/n$, with absorbing barriers at 0 and n. The worst-case expected time to converge is $n(n/2)^2 = n^3/4$, giving $\Pr[\text{time for } c \text{ to become linked} \geq \alpha n^3/8] \leq 2^{-\alpha}$ using the usual argument. Now apply the union bound to get $\Pr[\text{time for every } c \text{ to become linked} \geq \alpha n^3/8] \leq n2^{-\alpha}$ to get an expected coupling time of $O(n^3 \log n)$. In this case (say Aldous and Fill, quoting a of David Bruce Wilson [Wil04]) the bound is optimal up to a constant factor.

Real-world shuffling In real life, the handful of people who still use physical playing cards tend to use a **dovetail shuffle**, which is closely approximated by the reverse of a process where each card in a deck is independently assigned to a left or right pile and the left pile is place on top of the right pile. Coupling doesn't really help much here; instead, the process can be analyzed using more sophisticated techniques due to Bayer and Diaconis [BD92]; the short version of the result is that $\Theta(\log n)$ shuffles are needed to randomize a deck of size n.

7.3.5 Path coupling

Instead of just looking at X_t and Y_t, consider a path of intermediate states $X_t = Z_{0,t}Z_{1,t}Z_{2,t}\ldots Z_{m,t} = Y_t$, where each pair $Z_{i,t}Z_{i+1,t}$ is adjacent, i.e., satisfies $d(Z_{i,t}, Z_{i+1,t}) = 1$ for some reasonable metric for the state space. In choosing our metric, we will want to make sure that this is alway possible: that whenever two states are at distance d or less, there exists a path of $d-1$ intermediate states each separated by distance 1. The easiest way to do this is to define the metric based on path length after deciding which states we consider to be adjacent.

We now construct a coupling only for adjacent nodes that reduces their distance on average. The idea is that $d(X_t, Y_t) \leq \sum d(Z_{i,t}, Z_{i+1,t})$, so if the distance between each adjacent pair shrinks on average, so does the total length of the path. (If the path gets shorter, we can merge or drop some of the intermediate states, while still maintaining a path. If the path gets longer through some misfortune, we'll throw in a few new adjacent states to fill in the gaps.) When the length of the path reaches 0, we have $X_t = Y_t$. This technique is known as **path coupling**.

7.3.5.1 Sampling graph colorings

For example, let's look at sampling k-colorings of a graph with maximum degree d.[8] Consider the following chain on proper k-colorings of a graph with degree d, where k is substantially larger than d (we need at least $k \geq d+1$ to guarantee that a coloring exists, but we will assume k somewhat bigger to show fast convergence). At each step, we choose one of the n nodes and one of the k colors, and recolor the node to the new color if and only if no neighbor already has that color. (This process of changing one randomly-chosen state variable at each step to a new legal random value goes by the name of **Glauber dynamics** in statistical physics.)

Because $p_{ij} = p_{ji}$ for all pairs of states i, j, the detailed balance equations (7.2.1) hold when π_i is constant for all i, and we have a reversible Markov chain with a uniform stationary distribution. We'd like to show that we converge to this uniform distribution reasonably quickly, using a **path coupling** argument.

We'll think of colorings as vectors. Given two colorings x and y, let $d(x, y)$ be the Hamming distance between them, which is the number of nodes assigned different colorings by x and y. To show convergence, we will construct a coupling that shows that $d(X_t, Y_t)$ converges to 0 over time starting from arbitrary initial points X_0 and Y_0.

A complication for graph coloring is that it's not immediately evident that the length of the shortest path from X_t to Y_t is $d(X_t, Y_t)$. The problem is that it may not be possible to transform X_t into Y_t one node at a time without producing improper colorings. With enough colors, we can explicitly construct a short path between X_t and Y_t that uses only proper colorings; but for this particular process it is easier to simply extend the Markov chain to allow improper colorings, and show that our coupling works anyway. This also allows us to start with an improper coloring for X_0 if we are particularly lazy. The stationary distribution is not affected, because if j is an improper coloring, we have $p_{ij} = 0$ for all i, giving $\pi_j = 0$ (and also making the detailed balance equations continue to hold for improper colorings).

The natural coupling to consider given adjacent X_t and Y_t is to pick the same node and the same new color for both. If we pick the one node u on which they differ, and choose a color that is not used by any neighbor (which will be the same for both copies of the process, since all the neighbors have the same colors), then we get $X_{t+1} = Y_{t+1}$; this event occurs with probability

[8] The analysis here is weaker than it could be. See [MU05, §11.5] for a better analysis that only requires $2d$ colors, or [DGM02] for even more sophisticated results and a history of the problem.

at least $(k-d)/kn$. If we pick a node that is neither u nor adjacent to it, then the distance between X and Y doesn't change; either both get a new identical color or both don't. If we pick a node adjacent to u, then there is a possibility that the distance increases; if X_t assigns color c to u and Y_t color c', then choosing either c or c' for the neighbor will cause one copy to stay the same (because the new color isn't feasible) and the other to change. This event occurs with probability at most $2d/kn$ if we sum over all neighbors. So

$$\begin{aligned}\mathrm{E}[d(X_{t+1}, Y_{t+1})|d(X_t, Y_t) = 1] &\leq 1 - \frac{k-d}{kn} + \frac{2d}{kn} \\ &= 1 - \frac{k-3d}{kn}.\end{aligned}$$

This is less than 1 if $k > 3d$, or $k \geq 3d + 1$.

Applying this bound to the entire path, we get for general X_t, Y_t that

$$\begin{aligned}\mathrm{E}[d(X_{t+1}, Y_{t+1})|d(X_t, Y_t)] &\leq \left(1 - \frac{k-3d}{kn}\right) d(X_t, Y_t) \\ &\leq \exp\left(-\frac{k-3d}{kn}\right) d(X_t, Y_t).\end{aligned}$$

A simple induction then gives

$$\begin{aligned}\Pr[X_t \neq Y_t] &\leq \mathrm{E}[d(X_t, Y_t)] \\ &\leq \exp\left(-\frac{k-3d}{kn}t\right) \mathrm{E}[d(X_0, Y_0)] \\ &\leq \exp\left(-\frac{k-3d}{kn}t\right) n.\end{aligned}$$

Setting this bound equal to ϵ and solving for t gives

$$\tau(\epsilon) \leq \frac{kn}{k-3d} \ln(n/\epsilon).$$

7.3.5.2 Sampling independent sets

For a more complicated example of path coupling, let's try sampling independent sets of vertices on a graph $G = (V, E)$ with n vertices and m edges. If we can bias in favor of larger sets, we might even get a good independent set approximation! The fact that this is hard will console us when we find that it doesn't work.

By analogy with the graph coloring problem, a natural way to set up the random walk is to represent each potentially independent set as a bit vector, where 1 indicates membership in the set, and at each step we pick one of the n bits uniformly at random and set it to 0 or 1 with probability 1/2 each, provided that the resulting set is independent. (I.e., we'll use **Glauber dynamics** again.)

We can easily show that the stationary distribution of this process is uniform. The essential idea is that this is just a special case of random walk on a degree-d graph where we traverse each each with probability $\frac{1}{2d}$; in this case, we have $d = n$, since each independent set is adjacent to at most n other independent sets (obtained by flipping one of the bits).

It's also easy to see that $d(x, y) = \|x - y\|_1$ is a bound on the length of the minimum number of transitions to get from x to y, since we can alway remove all the extra ones from x and put back the extra ones in y while preserving independence throughout the process. This is a case where it may be hard to find the exact minimum path length, so we'll use this distance instead for our path coupling.

Here is an obvious coupling, that doesn't actually work: Pick the same position and value for both copies of the chain. If x and y are adjacent, then they coalesce with probability $1/n$ (both probability $1/2n$ transitions are feasible for both copies, since the neighboring nodes always have the same state). What is the probability that they diverge? We can only be prevented from picking a value if the value is 1 and some neighbor is 1. So the bad case is when $x_i = 1$, $y_i = 0$, and we attempt to set some neighbor of i to 1; in the worst case, this happens $d/2n$ of the time, which is at least $1/n$ when $d \geq 2$. No coalescence here!

We now have two choices. We can try to come up with a more clever random walk (see [MU05, §11.6] for a proof that a more sophisticated algorithm converges when $d \leq 4$), or we can try to persevere with our dumb random walk, a more clever analysis, and possibly a more restricted version of the problem where it will miraculously work even though it shouldn't really. Let's start with $d = 2$. Here the path coupling argument gives no expected change in $d(X_t, Y_t)$, so we have some hope that with enough wandering around they do in fact collide.

How do we prove this? Given arbitrary X_t, Y_t, we know from the path coupling argument above that on average they don't move any farther away. We also know that that there is a probability of at least $1/n$ that they move closer if they are not already identical (this is slightly tricky to see in the general case; basically we always have a $1/2n$ chance of removing an extra 1 from one or the other, and if we also have an extra 1 in the other process,

we get another $1/2n$, and if we don't, we can put in a missing 1 and get $1/2n$ that way instead). And we know that any change resulting from our not-very-smart coupling will change the distance by either 0 or ± 1. So if we sample only at times when a move has just occurred, we see a random walk (with a possible downward bias) with a reflecting barrier at n and an absorbing barrier at 0: this converges in n^2 steps. But since our random walk steps may take an expected n real steps each, we get a bound of n^3 on the total number of steps to converge.

But if $d = 3$, we seem to be doomed. We are also in trouble if we try to bias the walk in favor of adding vertices: since our good case is a $1 \to 0$ transition, decreasing its probability breaks the very weak balance we have even with $d = 2$ (for $d = 1$, it's not a problem, but this is an even less interesting case) So maybe we should see what we can do with the sneakier random walk describe in [MU05] (which is originally due to Luby and Vigoda [LV99]).

Here the idea is that we pick a random edge uv, and then try to do one of the following operations, all with equal probability:

1. Set $u = v = 0$.
2. Set $u = 0$ and $v = 1$.
3. Set $u = 1$ and $v = 0$.

In each case, if the result would be a non-independent set, we instead do nothing.

Verifying that this has a uniform stationary distribution is mildly painful if we are not careful, since there may be several different transitions that move from some state x to the same state y. But for each transition (occurring with probability $\frac{1}{3m}$), we can see that there is a reverse transition that occurs with equal probability; so the detailed balance equations (7.2.1) hold with uniform probabilities. Note that we can argue this even though we don't know what the actual stationary probabilities are, since we don't know how many independent sets our graph has.

So now what happens if we run two coupled copies of this process, where the copies differ on exactly one vertex i?

First, every neighbor of i is 0 in both processes. A transition that doesn't involve any neighbors of i will have the same effect on both processes. So we need to consider all choices of edges where one of the endpoints is either i or a neighbor j of i. In the case where the other endpoint isn't i, we'll call it k; there may be several such k.

If we choose ij and don't try to set j to one, we always coalesce the states. This occurs with probability $\frac{2}{3m}$. If we try to set i to zero and j to one, we may fail in both processes, because j may have a neighbor k that is already one; this will preserve the distance between the two processes. Similarly, if we try to set j to one as part of a change to some jk, we will also get a divergence between the two processes: in this case, the distance will actually increase. This can only happen if j has fewer than neighbors k (other than i) that are already in the independent set; if there are two such k, then we can't set j to one no matter what the state of i is.

This argument suggests that we need to consider three cases for each j, depending on the number s of nodes $k \neq i$ that are adjacent to j and have $x_k = y_k = 1$. In each case, we assume $x_i = 0$ and $y_i = 1$, and that all other nodes have the same value in both x and y.

- $s = 0$. Then if we choose ij, we can always set i and j however we like, giving a net $-\frac{1}{m}$ expected change to the distance. However, this is compensated for by up to $d - 1$ attempts to set $j = 1$ and $k = 0$ for some k, all of which fail in one copy of the process but succeed in the other. Since k doesn't change, each of these failures adds only 1 to the distance, which becomes at most $\frac{d-1}{3m}$ total. So our total expected change in this case is at most $\frac{d-4}{3m}$.

- $s = 1$. Here attempts to set $i = 0$ and $j = 1$ fail in both processes, giving only a $-\frac{2}{3m}$ expected change after picking ij. Any change to jk fails only if we set $j = 1$, which we can only do in the x process and only if we also set $k = 0$ for the unique k that is currently one. This produces an increase in the distance of 2 with probability $\frac{1}{3m}$, exactly canceling out the decrease from picking ij. Total expected change is 0.

- $s = 2$. Now we can never set $j = 1$. So we drop $-\frac{2}{3m}$ from changes to ij and have no change in distance from updates to jk for any $k \neq i$.

Considering all three cases, if $d \leq 4$, then in the worst case we have $\mathrm{E}[d(X_{t+1}, Y_{t+1}|X_t, Y_t] = d(X_t, Y_t)$. We also have that the distance changes with probability at least $\frac{2}{3m}$. So the same analysis as for the dumb process shows that we converge in at most $\frac{3}{8}n^2m$ steps on average.

Here, we've considered the case where all independent sets have the same probability. One can also bias the random walk in favor of larger independent sets by accepting increases with higher probability than decreases (as in Metropolis-Hastings); this samples independent sets of size s with probability proportional to λ^s. Some early examples of this approach are given

in [LV97, LV99, DG00]. The question of exactly which values of λ give polynomial convergence times is still open; see [MWW07] for some recent bounds.

7.3.5.3 Metropolis-Hastings and simulated annealing

Recall that the Metropolis-Hastings algorithm constructs a reversible Markov chain with a desired stationary distribution from any reversible Markov chain on the same states (see Section 7.2.2.2 for details.)

A variant, which generally involves tinkering with the chain while it's running, is the global optimization heuristic known as **simulated annealing**. Here we have some function g that we are trying to minimize. So we set $f(i) = \exp(-\alpha g(i))$ for some $\alpha > 0$. Running Metropolis-Hastings gives a stationary distribution that is exponentially weighted to small values of g; if i is the global minimum and j is some state with high $g(j)$, then $\pi(i) = \pi(j)\exp(\alpha(g(j) - g(i))$, which for large enough α goes a long way towards compensating for the fact that in most problems there are likely to be exponentially more bad j's than good i's. The problem is that the same analysis applies if i is a local minimum and j is on the boundary of some depression around i; large α means that it is exponentially unlikely that we escape this depression and find the global minimum.

The simulated annealing hack is to vary α over time; initially, we set α small, so that we get conductance close to that of the original Markov chain. This gives us a sample that is roughly uniform, with a small bias towards states with smaller $g(i)$. After some time we increase α to force the process into better states. The hope is that by increasing α slowly, by the time we are stuck in some depression, it's a deep one—optimal or close to it. If it doesn't work, we can randomly restart and/or decrease α repeatedly to jog the chain out of whatever depression it is stuck in. How to do this effectively is deep voodoo that depends on the structure of the underlying chain and the shape of $g(i)$, so most of the time people who use simulated annealing just try it out with some generic **annealing schedule** and hope it gives some useful result. (This is what makes it a heuristic rather than an algorithm. Its continued survival is a sign that it does work at least sometimes.)

Here are toy examples of simulated annealing with provable convergence times.

Single peak Let's suppose x is a random walk on an n-dimensional hypercube (i.e., n-bit vectors where we set 1 bit at a time), $g(x) = |x|$, and we

want to maximize g. Now a transition that increase $|x|$ is accepted always and a transition that decreases $|x|$ is accepted only with probability $e^{-\alpha}$. For large enough α, this puts a constant fraction of π on the single peak at $x = \mathbf{1}$; the observation is that that there are only $\binom{n}{k} \leq n^k$ points with k zeros, so the total weight of all points is at most $\pi(\mathbf{1}) \sum_{k \geq 0} n^k \exp(-\alpha k) = \pi(\mathbf{1}) \sum \exp(\ln n - \alpha)^k = \pi(\mathbf{1})/(1 - \exp(\ln n - \alpha)) = \pi(\mathbf{1}) \cdot O(1)$ when $\alpha > \ln n$, giving $\pi(\mathbf{1}) = \Omega(1)$ in this case.

So what happens with convergence? Let $p = \exp(-\alpha)$. Let's try doing a path coupling (see Section 7.3.5) between two adjacent copies x and y of the Metropolis-Hastings process, where we first pick a bit to change, then pick a value to assign to it, accepting the change in both processes if we can. The expected change in $|x - y|$ is then $(1/2n)(-1 - p)$, since if we pick the bit where x and y differ, we have probability $1/2n$ of setting both to 1 and probability $p/2n$ of setting both to 0, and if we pick any other bit, we get the same distribution of outcomes in both processes. This gives a general bound of $\mathrm{E}[|X_{t+1} - Y_{t+1}| \,|\, |X_t - Y_t|] \leq (1 - (1 + p)/2n) \, |X_t - Y_t|$, from which we have $\mathrm{E}[|X_t - Y_t|] \leq \exp(-t(1 + p)/2n) \, \mathrm{E}[|X_0 - Y_0|] \leq n \exp(-t(1 + p)/2n)$. So after $t = 2n/(1+p) \ln(n/\epsilon)$ steps, we expect to converge to within ϵ of the stationary distribution in total variation distance. This gives an $O(nlogn)$ algorithm for finding the peak. This is kind of a silly example, but in this case the running time for simulated annealing *is* almost equal to the $\Theta(n)$ optimum, if we scramble x by letting $g(x) = |x \oplus r|$ for some random constant r.

Single peak with very small amounts of noise Now we'll let $g : 2^n \to \mathbb{N}$ be some arbitrary Lipschitz function (in this case we are using the real definition: $|g(x) - g(y)| \leq |x - y|$) and ask for what values of $p = e^{-\alpha}$ the Metropolis-Hastings walk with $f(i) = e^{-\alpha g(i)}$ can be shown to converge quickly. Given adjacent states x and y, with $x_i \neq y_i$ but $x_j = y_j$ for all $j \neq i$, we still have a probability of at least $(1 + p)/2n$ of coalescing the states by setting $x_i = y_i$. But now there is a possibility that if we try to move to $(x[j/b], y[j/b])$ for some j and b, that x rejects while y does not or vice versa (note if $x_j = y_j = b$, we don't move in either copy of the process). Conditioning on j and b, this occurs with probability $1 - p$ precisely when $x[j/b] < x$ and $y[j/b] \geq y$ or vice versa, giving an expected increase in $|x - y|$ of $(1 - p)/2n$. We still get an expected net change of $-2p/2n = -p/n$ provided there is only one choice of j and b for which this

occurs. So we converge in time $\tau(\epsilon) \leq (n/p)\log(n/\epsilon)$ in this case.[9]

One way to think of this is that the shape of the neighborhoods of nearby points is similar. If I go up in a particular direction from point x, it's very likely that I go up in the same direction from some neighbor y of x.

If there are more bad choices for j and b, then we need a much larger value of p: the expected net change is now $(k(1-p)-1-p)/2n = (k-1-(k+1)p)/2n$, which is only negative if $p > (k-1)/(k+1)$. This gives much weaker pressure towards large values of g, which still tends to put us in high neighborhoods but creates the temptation to fiddle with α to try to push us even higher once we think we are close to a peak.

7.4 Spectral methods for reversible chains

(See also `http://www.stat.berkeley.edu/~aldous/RWG/Chap3.pdf`, from which many of the details in the notes below are taken.)

The problem with coupling is that (a) it requires cleverness to come up with a good coupling; and (b) in many cases, even that doesn't work—there are problems for which no coupling that only depends on current and past transitions coalesces in a reasonable amount of time.[10] When we run into these problems, we may be able to show convergence instead using a linear-algebraic approach, where we look at the eigenvalues of the transition matrix of our Markov chain. This approach works best for reversible Markov chains, as described in the next section.

7.4.1 Time-reversed chains

Given any Markov chain with transition matrix P and stationary distribution π, define the **time-reversed chain** with matrix P^* where $\pi_i p_{ij} = \pi_j p^*_{ji}$. (Check: $\sum_i p^*_{ji} = \sum_i p_{ij}\pi_i/\pi_j = \pi_j/\pi_j = 1$, so it's a Markov chain, and the fact P^*'s stationary distribution is the same as P's, and that in general P^*'s paths starting from the stationary distribution are a reverse of P's paths starting from the same distribution follows from an argument similar to that for reversible chains.) This gives an alternate definition of a reversible chain as a chain for which $P = P^*$.

Examples:

[9] You might reasonably ask if such functions g exist. One example is $g(x) = (x_1 \oplus x_2) + \sum_{i \geq 2} x_i$.

[10] Such a coupling is called a **causal coupling**; an example of a Markov chain for which causal couplings are known not to work is one used for sampling perfect matchings in bipartite graphs described in Section 7.4.6.4 [KR99].

- Given a biased random walk on a cycle that moves right with probability p and left with probability q, its time-reversal is the walk that moves left with probability p and right with probability q. (Here the fact that the stationary distribution is uniform makes things simple.)

- Given the random walk defined by $X_{t+1} = X_t + 1$ with probability 1/2 and 0 with probability 1/2, we have $\pi_i = 2^{-i-1}$. This is not reversible (there is a transition from 1 to 2 but none from 2 to 1), but we can reverse it by setting $p^*_{ij} = 1$ for $i = j + 1$ and $p^*_{0i} = 2^{-i-1}$. (Check: $\pi_i p_{ii+1} = 2^{-i-1}(1/2) = \pi_{i+1} p^*_{i+1i} = 2^{-i-2}(1); \pi_i p_{i0} = 2^{-i-1}(1/2) = \pi_0 p^*_{0i} = (1/2)2^{-i-1}$.)

Reversed versions of chains with messier stationary distributions are messier.

We can use time-reversal to generate reversible chains from arbitrary chains. The chain with transition matrix $(P + P^*)/2$ (corresponding to moving 1 step forward or back with equal probability at each step) is always a reversible chain.

7.4.2 Spectral properties of a reversible chain

(See `http://www.stat.berkeley.edu/~aldous/RWG/Chap3.pdf`, Section 4, or the lecture notes from Ravi Montenegro at `http://www.ravimontenegro.com/8843/notes/lecture5.pdf`.)

Suppose P is irreducible and reversible, i.e. that $\pi_i \neq 0$ for all i and $\pi_i p_{ij} = \pi_j p_{ji}$ for all i, j. Divide both sides by $\sqrt{\pi_i \pi_j}$ to get $(\pi_i)^{1/2} p_{ij} (\pi_j)^{-1/2} = (\pi_j)^{1/2} p_{ji} (\pi_i)^{-1/2}$. This shows that the matrix S with entries $S_{ij} = (\pi_i)^{1/2} p_{ij} (\pi_j)^{-1/2}$ is symmetric.

Why we care about this: Any symmetric real-valued matrix has can, by the **spectral theorem**[11] be decomposed as $S = U \Lambda U'$ where Λ is a diagonal matrix whose entries are eigenvalues of S and the columns of U are orthonormal eigenvectors. If you didn't take the advice to take a linear algebra course as part of your undergraduate Computer Science major, an eigenvector of a matrix is a vector v such that $vS = \lambda v$, where λ is the corresponding eigenvalue—in other words, it's a vector that changes in length but not direction when we run it through S. The fact that the columns of U are orthonormal (orthogonal to each other, and having length 1), means that they act like a set of coordinate axes and that vU gives the coordinates of v

[11] See `http://en.wikipedia.org/wiki/Spectral_theorem`.

in this coordinate system, while $(vU)U'$ undoes the transformation by multiplying each coordinate by the corresponding row vector, since UU' equals the identity matrix I.

In more detail, we can write an arbitrary row vector $v = c_1u_1 + c_2u_2 + \ldots c_nu_n$, where the vectors u_i correspond to the rows of U' and the coefficients c_i are unique. If we multiply v by S on the right, we get $vU\Lambda U' = \sum \lambda_i c_i u_i$, and if we multiply it by S^t, we get $v(U\Lambda U')^t = vU\Lambda^t U' = \sum \lambda_i^t c_i u_i$. So repeatedly multiplying by S corresponds to shrinking v along the eigenvectors with eigenvalues whose absolute value is less than 1, while leaving behind the component(s) corresponding to eigenvalues 1 or greater in absolute value.

For example, the symmetric transition matrix

$$S = \begin{bmatrix} p & q \\ q & p \end{bmatrix}$$

corresponding to a Markov chain on two states that stays in the same state with probability p and switches to the other state with probability $q = 1-p$ has eigenvectors (not normalized) $\begin{bmatrix} 1 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & -1 \end{bmatrix}$ with corresponding eigenvalues 1 and $p - q$, as shown by computing

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} p & q \\ q & p \end{bmatrix} = \begin{bmatrix} p+q & q+p \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 & 1 \end{bmatrix}$$

and

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} p & q \\ q & p \end{bmatrix} = \begin{bmatrix} p-q & q-p \end{bmatrix} = (p-q) \cdot \begin{bmatrix} 1 & -1 \end{bmatrix}.$$

This gives a spectral decomposition

$$S = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & p-q \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix},$$

where the $1/\sqrt{2}$ factors are included to make all the eigenvectors have length 1, which implies that repeatedly multiplying some initial vector x by S preserves the sum of x_1 and x_2 while multiplying the difference $x_0 - x_1$ by $p - q$ at each step.

What if we multiply v by P^t, where P is the (not necessarily symmetric) transition matrix of a reversible Markov chain? Let Π be the diagonal matrix given by $\Pi_{ii} = \sqrt{\pi_i}$; then we can let $S = \Pi P \Pi^{-1}$ and thus $P = \Pi^{-1} S \Pi$, where S is a symmetric matrix (as shown at the beginning of this section). So

$P^t = (\Pi^{-1} S \Pi)^t = \Pi^{-1} S^t \Pi = \Pi^{-1} U \Lambda^t U' \Pi$, where $S = U \Lambda U'$ is the spectral decomposition of S. So we can compute vP^t by dividing each component of v by the square root of the stationary distribution, shrinking along all the eigenvectors, and scaling back. If we expand out all the matrix products to get a specific $(vP^t)_{ij}$, we obtain the **spectral representation formula**:

$$\Pr[X_t = j | X_0 = i] = \sqrt{\frac{\pi_j}{\pi_i}} \sum_{m=1}^{n} \lambda_m^t u_{mi} u_{mj}.$$

Assuming that P is irreducible and aperiodic, we have a unique limit to this process. This means that S has exactly one eigenvalue 1 (corresponding to the eigenvector whose entries are $(\pi_i)^{1/2}$), and all the remaining eigenvalues λ satisfy $\lambda^t \to 0$ as $t \to \infty$. We can write the eigenvalues of S (which turn out to also be the eigenvalues of P, although the eigenvectors are different) as $1 = \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > -1$.[12] Assuming that $|\lambda_2| \geq |\lambda_n|$, as t grows large λ_2^t will dominate the other smaller eigenvalues, and so the size of λ_2 will control the rate of convergence of the underlying Markov process.

In detail, we have that the first eigenvector u_1 satisfies $u_{1i} = \sqrt{\pi_i}$; so the first term in the spectral representation formula for $\Pr[X_t = j | X_0 = i]$ is $\left(\sqrt{\pi_j/\pi_i}\right) 1^t \left(\sqrt{\pi_i \pi_j}\right) = \pi_j$. Each subsequent term is $\sqrt{\pi_j/\pi_i}(\lambda_m)^t u_{mi} u_{mj} \leq \sqrt{\pi_j/\pi_i}(\lambda_m)^t$, giving a bound of $(n-1)\sqrt{\pi_j/\pi_i}(\lambda_{\max})^t$ where $\lambda_{\max} = \max(\lambda_2, |\lambda_n|)$ on the total (in the unlikely event that all the eigenvalues except λ_1 are close to $\lambda_{\max}$) and $O(\sqrt{\pi_i/\pi_j}(\lambda_{\max})^t)$ in the typical case. In either case, since $\lambda_{\max} < 1$, we get an exponentially-decreasing bound.

It is often convenient to express this bound as $\exp(-t/c)$ for an appropriate constant. Here we observe that $\lambda_{\max} = 1 - (1 - \lambda_{\max}) \leq \exp(-(1 - \lambda_{\max}))$ gives $(\lambda_{\max})^t \leq \exp(-(1 - \lambda_{\max})t)$. It is customary to describe this fact by giving the **mixing rate** or **relaxation time** $\tau_2 = 1/(1 - \lambda_{\max})$, this being the time for which this bound drops by a factor of $1/e$.

So far we have just considered the probability that we land in a particular state. To bound total variation distance, we (following the Montenegro notes mentioned above) go through the L_p norm. Given a vector v, define $\|v\|_{p,\pi} = (\sum_i |v_i|^p \pi_i)^{1/p}$. (This is a weighted version of the usual L_p norm.) Then $d_{TV}(p, \pi) = (1/2) \|\pi - p\|_1 = (1/2) \|1 - p/\pi\|_{1,\pi}$. A general property

[12] For aperiodic chains and chains with period 2, all the eigenvalues are real; the period-2 case includes an eigenvalue of -1. Chains with periods greater than 2 have pairs of complex-valued eigenvalues that are roots of unity, which happen to cancel out to only produce real probabilities in vP^t. Chains that aren't irreducible will have one eigenvector with eigenvalue 1 for each final component; the stationary distributions of these chains are linear combinations of these eigenvectors (which are just the stationary distributions on each component).

of L_p norms is that if we increase p, we also increase $\|v\|_{p,\pi}$, so we can bound $d_{TV}(p,\pi)$ by $(1/2)\|1 - p/\pi\|_{2,\pi}$, which is a little easier to compute. Omitting the details of the proof (see `http://www.ravimontenegro.com/8843/notes/lecture5.pdf`), the result is that, starting from an initial vector x that puts all of its weight on state i,

$$d_{TV}(xP^t, \pi) \leq \frac{1}{2}\lambda_{\max}^t \sqrt{\frac{1-\pi_i}{\pi_i}}.$$

So now we just need a tool for bounding $\lambda_{\max}$.

7.4.3 Conductance

The **conductance** $\Phi(S)$ of a set S is

$$\Phi(S) = \frac{\sum_{i\in S, j\notin S} \pi_i p_{ij}}{\pi(S)}. \tag{7.4.1}$$

This is the probability of leaving S on the next step starting from the stationary distribution conditioned on being in S. The conductance is a measure of how easy it is to escape from a set; it can also be thought of as a weighted version of edge expansion.

The conductance of a Markov chain as a whole is obtained by taking the minimum of $\Phi(S)$ over all S that occur with probability at most $1/2$:

$$\Phi = \min_{0<\pi(S)\leq 1/2} \Phi(S). \tag{7.4.2}$$

The usefulness of conductance is that it bounds λ_2:

Theorem 7.4.1. *In a reversible Markov chain,*

$$1 - 2\Phi \leq \lambda_2 \leq 1 - \Phi^2/2. \tag{7.4.3}$$

This result is a generalization of the **Cheeger inequality** from graph theory, and was proved for Markov chains by Jerrum and Sinclair [JS89]. We won't attempt to prove it here.

For lazy walks we always have $\lambda_2 = \lambda_{\max}$, and so we can convert this to a bound on the relaxation time:

Corollary 7.4.2.

$$\frac{1}{2\Phi} \leq \tau_2 \leq \frac{2}{\Phi^2}. \tag{7.4.4}$$

In other words, high conductance implies low relaxation time and vice versa, up to squaring.

For very simple Markov chains we can compute the conductance directly. Consider a lazy random walk on an odd cycle. Any proper subset S has at least two outgoing edges, each of which carries a flow of $1/4n$, giving $\Phi_S \geq (1/2n)/\pi(S)$. If we now take the minimum of Φ_S over all S with $\pi(S) \leq 1/2$, we get $\Phi \geq 1/n$, which gives $\tau_2 \leq 2n^2$. This is essentially the same bound as we got from coupling.

Here's a slightly more complicated chain. Take two copies of K_n, and join them by a path with n edges. Now consider Φ_S for a lazy random walk on this graph where S consists of half the graph, split in the middle of the path (we can't actually do this exactly, but we'll get as close as we can). There is a single outgoing edge uv, with $\pi(u) = d(u)/2\left|E\right| = 2/(2n(n-1)n/2+n) = 2n^{-2}$ and $p_{uv} = 1/4$, for $\pi(u)p_{uv} = n^{-2}/2$. By symmetry, we have $\pi(S) \to 1/2$ as $n \to \infty$, giving $\Phi_S \to n^{-2}(1/2)/(1/2) = n^{-2}$. So we have $n^2/2 \leq \tau_2 \leq 2n^4$.

How does this compare to the actual mixing time? In the stationary distribution, we have a constant probability of being in each of the copies of K_n. Suppose we start in the left copy. At each step there is a $1/n$ chance that we are sitting on the path endpoint. We then step onto the path with probability $1/n$, and reach the other end before coming back with probability $1/n$. So (assuming we can make this extremely sloppy handwaving argument rigorous) it takes at least n^3 steps on average before we reach the other copy of K_n, which gives us a rough estimate of the mixing time of $\Theta(n^3)$. In this case the exponent is exactly in the middle of the bounds derived from conductance.

7.4.4 Edge expansion using canonical paths

(Here and below we are mostly following the presentation in [Gur00], except with slightly different examples and substantially more errors.)

For more complicated Markov chains, it is helpful to have a tool for bounding conductance that doesn't depend on intuiting what sets have the smallest boundary. The **canonical paths** [JS89] method does this by assigning a unique path γ_{xy} from each state x to each state y in a way that doesn't send too many paths across any one edge. So if we have a partition of the state space into sets S and T, then there are $|S| \cdot |T|$ paths from states in S to states in T, and since (a) every one of these paths crosses an S–T edge, and (b) each S–T edge carries at most ρ paths, there must be at least $|S| \cdot |T| / \rho$ edges from S to T. Note that because there is no guarantee we

chose good canonical paths, this is only useful for getting lower bounds on conductance—and thus upper bounds on mixing time—but this is usually what we want.

Let's start with a small example. Let $G = C_n \square C_m$, the $n \times m$ torus. A lazy random walk on this graph moves north, east, south, or west with probability 1/8 each, wrapping around when the coordinates reach n or m. Since this is a random walk on a regular graph, the stationary distribution is uniform. What is the relaxation time?

Intuitively, we expect it to be $O(\max(n, m)^2)$, because we can think of this two-dimensional random walk as consisting of two one-dimensional random walks, one in the horizontal direction and one in the vertical direction, and we know that a random walk on a cycle mixes in $O(n^2)$ time. Unfortunately, the two random walks are not independent: every time I take a horizontal step is a time I am not taking a vertical step. We *can* show that the expected coupling time is $O(n^2 + m^2)$ by running two sequential instances of the coupling argument for the cycle, where we first link the two copies in the horizontal direction and then in the vertical direction. So this gives us one bound on the mixing time. But what happens if we try to use conductance?

Here it is probably easiest to start with just a cycle. Given points x and y on C_n, let the canonical path γ_{xy} be a shortest path between them, breaking ties so that half the paths go one way and half the other. Then each each is crossed by exactly k paths of length k for each $k = 1 \dots (n/2 - 1)$, and at most $n/4$ paths of length $n/2$ (0 if n is odd), giving a total of $\rho \leq (n/2 - 1)(n/2)/2 + n/4 = n^2/8$ paths across the edge.

If we now take an S–T partition where $|S| = m$, we get at least $m(n - m)/\rho = 8m(n - m)/n^2$ S–T edges. This peaks at $m = n/2$, where we get 2 edges—exactly the right number—and in general when $m \leq n/2$ we get at least $8m(n/2)/n^2 = 4m/n$ outgoing edges, giving a conductance $\Phi_S \geq (1/4n)(4m/n)/(m/n) = 1/n$. (This is essentially what we got before, except we have to divide by 2 because we are doing a lazy walk. Note that for small m, the bound is a gross underestimate, since we know that every nonempty proper subset has at least 2 outgoing edges.)

Now let's go back to the torus $C_n \square C_m$. Given x and y, define γ_{xy} to be the L-shaped path that first changes x_1 to y_1 by moving the shortest distance vertically, then changes x_2 to y_2 by moving the shortest distance horizontally. For a vertical edge, the number of such paths that cross it is bounded by $n^2 m/8$, since we get at most $n^2/8$ possible vertical path segments and for each such vertical path segment there are m possible horizontal destinations to attach to it. For a horizontal edge, n and m are reversed, giving $m^2 n/8$

paths. To make our life easier, let's assume $n \geq m$, giving a maximum of $\rho = n^2m/8$ paths.

For $|S| \leq nm/2$, this gives at least $|S| \cdot |G \setminus S| / \rho \geq |S| (nm/2)/(n^2m/8) = 4|S|/n$ outgoing edges. We thus have $\phi_S \geq (1/8nm)(4|S|/n)/(|S|/nm) = 1/2n$. This gives $\tau_2 \leq 2/\Phi^2 \leq 8n^2$, pretty much what we expect.

7.4.5 Congestion

For less symmetrical chains, we weight paths by the probabilities of their endpoints when counting how many cross each edge, and treat the flow across the edge as a capacity. This gives the **congestion** of a collection of canonical paths $\Gamma = \{\gamma_{xy}\}$, which is computed as

$$\rho(\Gamma) = \max_{uv \in E} \frac{1}{\pi_u p_{uv}} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y,$$

and we define $\rho = \min_\Gamma \rho(\Gamma)$.

Congestion is useful because of the following lemma:

Lemma 7.4.3.

$$\Phi \geq \frac{1}{2\rho} \tag{7.4.5}$$

from which it follows that

$$\tau_2 \leq 8\rho^2. \tag{7.4.6}$$

Proof. Pick some set of canonical paths Γ with $\rho(\Gamma) = \rho$. Now pick some S–T partition with $\Phi(S) = \Phi$. Consider a flow where we route $\pi(x)\pi(y)$ units of flow along each path γ_{xy} with $x \in S$ and $y \in T$. This gives a total flow of $\pi(S)\pi(T) \geq \pi(S)/2$. We are going to show that we need a lot of capacity across the S–T cut to carry this flow, which will give the lower bound on conductance.

For any S–T edge uv, we have

$$\frac{1}{\pi_u p_{uv}} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y \leq \rho$$

or

$$\sum_{\gamma_{xy} \ni uv} \pi_x \pi_y \leq \rho \pi_u p_{uv}.$$

Since each S–T path crosses at least one S–T edge, we have

$$\begin{aligned}\pi(S)\pi(T) &= \sum_{x\in S, y\in T} \pi_x \pi_y \\ &\le \sum_{u\in S, t\in V, uv\in E} \sum_{\gamma_{xy}\ni uv} \pi_x \pi_y \\ &\le \sum_{u\in S, t\in V, uv\in E} \rho \pi_u p_{uv}. \\ &= \rho \sum_{u\in S, t\in V, uv\in E} \pi_u p_{uv}.\end{aligned}$$

But then

$$\begin{aligned}\Phi(S) &= \frac{\sum_{u\in S, t\in V, uv\in E} \pi_u p_{uv}}{\pi_S} \\ &\ge \frac{\pi(S)\pi(T)/\rho}{\pi(S)} \\ &\ge \frac{1}{2\rho}.\end{aligned}$$

To get the bound on τ_2, use (7.4.4) to compute $\tau_2 \le 2/\phi^2 \le 8\rho^2$. □

7.4.6 Examples

7.4.6.1 Lazy random walk on a line

Consider a lazy random walk on a line with reflecting barriers at 0 and $n-1$. Here $\pi_x = \frac{1}{n}$ for all x. There aren't a whole lot of choices for canonical paths; the obvious choice for γ_{xy} with $x < y$ is $x, x+1, x+2, \ldots, y$. This puts $(n/2)^2$ paths across the middle edge (which is the most heavily loaded), each of which has weight $\pi_x \pi_y = n^{-2}$. So the congestion is $\frac{1}{(1/n)(1/4)}(n/2)^2 n^{-}2 = n$, giving a mixing time of at most $8n^2$. This is a pretty good estimate.

7.4.6.2 Random walk on a hypercube

Let's try a more complicated example: the random walk on a hypercube from Section 7.3.3. Here at each step we pick some coordinate uniformly at random and set it to 0 or 1 with equal probability; this gives a transition probability $p_{uv} = \frac{1}{2n}$ whenever i and j differ by exactly one bit. A plausible choice for canonical paths is to let γ_{xy} use **bit-fixing routing**, where we change bits in x to the corresponding bits in y from left to right. To compute

congestion, pick some edge uv, and let k be the bit position in which u and v differ. A path γ_{xy} will cross uv if $u_k \dots u_n = x_k \dots x_n$ (because when we are at u we haven't fixed those bits yet) and $v_1 \dots v_k = y_1 \dots y_k$ (because at v we have fixed all of those bits). There are 2^{k-1} choices of $x_1 \dots x_{k-1}$ consistent with the first condition and 2^{n-k} choices of $y_{k+1} \dots y_n$ consistent with the second, giving exactly 2^{n-1} total paths γ_{xy} crossing uv. Since each path occurs with weight $\pi_x \pi_y = 2^- 2n$, and the flow across uv is $\pi_u p_{uv} = 2^{-n} \frac{1}{2n}$, we can calculate the congestion

$$\begin{aligned}\rho(\Gamma) &= \max_{uv \in E} \frac{1}{\pi_u p_{uv}} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y \\ &= \frac{1}{2^{-n}/2n} \cdot 2^{n-1} \cdot 2^{-2n} \\ &= n.\end{aligned}$$

This gives a relaxation time $\tau_2 \leq 8\rho^2 = 8n^2$. In this case the bound is substantially worse than what we previously proved using coupling.

The fact that the number of canonical paths that cross a particular edge is exactly 1/2 the number of nodes in the hypercube is not an accident: if we look at what information we need to fill in to compute x and y from u and v, we need (a) the part of x we've already gotten rid of, plus (b) the part of y we haven't filled in yet. If we stitch these two pieces together, we get all but one of the n bits we need to specify a node in the hypercube, the missing bit being the bit we flip to get from u to v. This sort of thing shows up a lot in conductance arguments where we build our canonical paths by fixing a structure one piece at a time.

7.4.6.3 Matchings in a graph

A **matching** in a graph $G = (V, E)$ is a subset of the edges with no two elements adjacent to each other; equivalently, it's a subgraph that includes all the vertices in which each vertex has degree at most 1. We can use a random walk to sample matchings from an arbitrary graph uniformly.

Here is the random walk. Let S_t be the matching at time t. At each step, we choose an edge $e \in E$ uniformly at random, and flip a coin to decide whether to include it or not. If the coin comes up tails, set $S_{t+1} = S_t \setminus \{e\}$ (this may leave $S_{t+1} = S_t$ if S_t already omitted e); otherwise, set $S_{t+1} = S_t \cup \{e\}$ unless one of the endpoints of e is already incident to an edge in S_t, in which case set $S_{t+1} = S_t$.

Because this chain contains many self-loops, it's aperiodic. It's also straightforward to show that any transition between two adjacent matchings

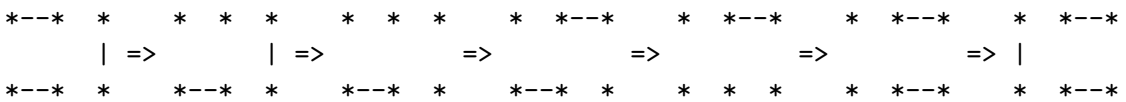
occurs with probability exactly $\frac{1}{2|E|}$, and thus that the chain is reversible with a uniform stationary distribution. We'd like to bound the congestion of the chain to show that it converges in a reasonable amount of time.

Let N be the number of matchings in G, and let $n = |V|$ and $m = |E|$ as usual. Then $\pi_S = 1/N$ for all S and $\pi_S p_{ST} = \frac{1}{2Nm}$. Our congestion for any transition ST will then be $2NmN^{-2} = 2m/N$ times the number of paths that cross ST; ideally this number of paths will be at most N times some small polynomial in n and/or m.

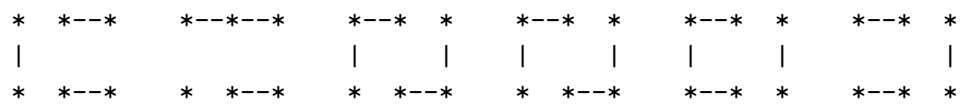
Suppose we are trying to get from some matching X to another matching Y. The graph $X \cup Y$ has maximum degree 2, so each of its connected components is either a path or a cycle; in addition, we know that the edges in each of these paths or cycles alternate whether they come from X or Y, which among other things implies that the cycles are all even cycles.

We can transform X to Y by processing these components in a methodical way: first order the components (say, by the increasing identity of the smallest vertex in each), then for each component replace the X edges with Y edges. To do the replacement, we walk across the path or cycle adding Y edges in some fixed order after first deleting the adjacent X edges. If we are clever about choosing which Y edge to start with (basically always pick an edge with only one adjacent X edge if possible, and then use edge ordering to break ties) we can ensure that for each transition ST it holds $(X \cap Y) \setminus (S \cap T)$ consists of a matching plus an extra edge.

Here is an ASCII art version of this process applied to a cycle of length 6:



We now want to show that not too many of these canonical paths use any particular transition. The trick is that when we do a transition from S to T, the graph $(X \cap Y) \setminus (S \cap T)$ is *almost* a matching:



In each case we can turn the $(X \cap Y) \setminus (S \cap T)$ into a matching by deleting one edge. So we can compute $X \cap Y$ knowing $S \cap T$ by supplying a matching ($\leq N$ choices) plus an extra edge ($\leq m$ choices). Once we know what $X \cap Y$ is, we can reconstruct X and Y by looking at which component

we are on, assigning the edges in the earlier components to Y (because we must have done them already), assigning the edges in the later components to X (because we haven't done them yet), splitting the edges in our current component appropriately (because we know how far we've gotten), and then taking complements as needed with respect to $X \cap Y$ to get the remaining edges for each of X and Y. Since there are at most Nm ways to do this, we get that the ST transition is covered by at most Nm canonical paths γ_{XY}.

Plugging this back into our previous formula, we get $\rho \leq (2m/N)(Nm) = m^2$, which gives $\tau_2 \leq 8m^4$.

7.4.6.4 Perfect matchings in dense bipartite graphs

(Basically doing [MR95, §11.3].)

A similar chain can be used to sample perfect matchings in dense bipartite graphs, as shown by Jerrum and Sinclair [JS89] based on an algorithm by Broder [Bro86] that turned out to have a bug in the analysis [Bro88].

A **perfect matching** of a graph G is a subgraph that includes all the vertices and gives each of them exactly one incident edge. We'll be looking for perfect matchings in a **bipartite graph** consisting of n left vertices $u_1, \ldots, u_n$ and n right vertices $v_1, \ldots, v_n$, where every edge goes between a left vertex and a right vertex. We'll also assume that the graph is **dense**, which in this case means that every vertex has at least $n/2$ neighbors. This density assumption was used in the original Broder and Jerrum-Sinclair papers but removed in a later paper by Jerrum, Sinclair, and Vigoda [JSV04].

The random walk is similar to the random walk from the previous section restricted to the set of matchings with either $n-1$ or n edges. At each step, we first flip a coin (the usual lazy walk trick); if it comes up heads, we choose an edge uv uniformly at random and apply one of the following transformations depending on how uv fits in the current matching m_t:

1. If $uv \in m_t$, and $|m_t| = n$, set $m_{t+1} = m_t \setminus \{uv\}$.
2. If u and v are both unmatched in m_t, and $|m_t| = n-1$, set $m_{t+1} = m_t \cup \{uv\}$.
3. If exactly one of u and v is matched to some other node w, and $|m_t| = n-1$, perform a rotation that deletes the w edge and adds uv.
4. If none of these conditions hold, set $m_{t+1} = m_t$.

The walk can be started from any perfect matching, which can be found in $O(n^{5/2})$ time using a classic algorithm of Hopcroft and Karp [HK73]

that repeatedly searches for an **augmenting path**, which is a path in G between two unmatched vertices that alternates between edges not in the matching and edges in the matching. (We'll see augmenting paths again below when we show that any near-perfect matching can be turned into a perfect matching using at most two transitions.)

We can show that this walk converges in polynomial time using a canonical path argument. This is done in two stages: first, we define canonical paths between all perfect matchings. Next, we define a short path from any matching of size $n-1$ to some nearby perfect matching, and build paths between arbitrary matchings by pasting one of these short paths on one or both ends of a long path between perfect matchings. This gives a collection of canonical paths that we can show to have low congestion.

To go between perfect matchings X and Y, consider $X \cap Y$ as a collection of paths and even cycles as in Section 7.4.6.3. In some standard order, fix each path cycle by first deleting one edge to make room, then using rotate operations to move the rest of the edges, then putting the last edge back in. For any transition S–T along the way, we can use the same argument that we can compute $X \cap Y$ from $S \cap T$ by supplying the missing edges, which will consist of a matching of size n plus at most one extra edge. So if N is the number of matchings of size n or $n-1$, the same argument used previously shows that at most Nm of these long paths cross any S–T transition.

For the short paths, we must use the density property. The idea is that for any matching that is not perfect, we can find an augmenting path of length at most 3 between two unmatched nodes on either side. Pick some unmatched nodes u and v. Each of these nodes is adjacent to at least $n/2$ neighbors; if any of these neighbors are unmatched, we just found an augmenting path of length 1. Otherwise the $n/2$ neighbors of u and the $n/2$ nodes matched to neighbors of v overlap (because v is unmatched, leaving at most $n-1$ matched nodes and thus at most $n/2-1$ nodes that are matched to something that's not a neighbor of v). So for each matching of size $n-1$, in at most two steps (rotate and then add an edge) we can reach some specific perfect matching. There are at most m^2 ways that we can undo this, so each perfect matching is associated with at most m^2 smaller matchings. This blows up the number of canonical paths crossing any transition by roughly m^4; by counting carefully we can thus show congestion that is $O(m^6)$ ($O(m^4)$ from the blow-up, m from the m in Nm, and m from $1/p_{ST}$).

It follows that for this process, $\tau_2 = O(m^1 2)$. (I think a better analysis is possible.)

As noted earlier, this is an example of a process for which causal coupling doesn't work in less than exponential time [KR99], a common problem with

Markov chains that don't have much symmetry. So it's not surprising that stronger techniques were developed specifically to attack this problem.

Chapter 8

Approximate counting

(See [MR95, Chapter 11] for details.)

Basic idea: we have some class of objects, we want to know how many of them there are. Ideally we can build an algorithm that just prints out the exact number, but for many problems this is hard.

A **fully polynomial randomized approximation scheme** or **FPRAS** for a numerical problem outputs a number that is between $(1-\epsilon)$ and $(1+\epsilon)$ times the correct answer, with probability at least $3/4$ (or some constant bounded away from $1/2$—we can amplify to improve it), in time polynomial in the input size n and $(1/\epsilon)$. In this chapter, we'll be hunting for FPRASs. But first we will discuss briefly why we can't just count directly in many cases.

8.1 Exact counting

A typical application is to a problem in the complexity class #**P**, problems that involve counting the number of accepting computation paths in a nondeterministic polynomial-time Turing machine. Equivalently, these are problems that involve counting for some input x the number of values r such that $M(x, r) = 1$, where M is some machine in **P**. An example would be the problem **#SAT** of counting the number of satisfying assignments of some CNF formula.

The class #**P** (which is usually pronounced **sharp P** or **number P**) was defined by Leslie Valiant in a classic paper [Val79]. The central result in this paper is **Valiant's theorem**. This shows that any problem in #**P** can be **reduced** (by **Cook reductions**, meaning that we are allowed to use the target problem as a subroutine instead of just calling it once) to the

problem of computing the **permanent** of a square 0–1 matrix A, where the permanent is given by the formula $\sum_\pi \prod_i A_{i,\pi(i)}$, where the sum ranges over all $n!$ permutations π of the indices of the matrix. An equivalent problem is counting the number of **perfect matchings** (subgraphs including all vertices in which every vertex has degree exactly 1) of a bipartite graph. Other examples of #**P**-complete problems are #SAT (defined above) and **#DNF** (like #SAT, but the input is in DNF form; #SAT reduces to #DNF by negating the formula and then subtracting the result from 2^n).

Exact counting of #**P**-hard problems is likely to be very difficult: **Toda's theorem** [Tod91] says that being able to make even a single query to a #**P**-oracle is enough to solve any problem in the **polynomial-time hierarchy**, which contains most of the complexity classes you have probably heard of. Nonetheless, it is often possible to obtain good approximations to such problems.

8.2 Counting by sampling

If many of the things we are looking for are in the target set, we can count by sampling; this is what poll-takers do for a living. Let U be the universe we are sampling from and G be the "good" set of points we want to count. Let $\rho = |G| / |U|$. If we can sample uniformly from U, then we can estimate ρ by taking N independent samples and dividing the number of samples in G by N. This will give us an answer $\hat{\rho}$ whose expectation is ρ, but the accuracy may be off. Since the variance of each sample is $\rho(1-\rho) \approx \rho$ (when ρ is small), we get $\operatorname{Var}[\sum X_i] \approx m\rho$, giving a standard deviation of $\sqrt{m\rho}$. For this to be less than our allowable error $\epsilon m\rho$, we need $1 \leq \epsilon\sqrt{m\rho}$ or $N \geq \frac{1}{\epsilon^2\rho}$. This gets bad if ρ is exponentially small. So if we are to use sampling, we need to make sure that we only use it when ρ is large.

On the positive side, if ρ is large enough we can easily compute how many samples we need using Chernoff bounds. The following lemma gives a convenient estimate; it is based on [[MR95, Theorem 11.1]] with a slight improvement on the constant:

Lemma 8.2.1. *Sampling N times gives relative error ϵ with probability at least $1-\delta$ provided $\epsilon \leq 1.81$ and*

$$N \geq \frac{3}{\epsilon^2\rho} \ln \frac{2}{\delta}. \tag{8.2.1}$$

Proof. Suppose we take N samples, and let X be the total count for these samples. Then $\mathrm{E}[X] = \rho N$, and combining (3.2.3) and (3.2.7) we have, for

$\epsilon \leq 4.11$,

$$\Pr[|X - \rho N| \geq \epsilon \rho N] \leq 2e^{-\rho N \epsilon^2/3}.$$

Now set $2e^{\rho N \epsilon^2/3} \leq \delta$ and solve for N to get (8.2.1). □

8.3 Approximating #DNF

(Basically just repeating presentation in [MR95, §11.2] of covering technique from Karp and Luby [KL85].)

A **DNF formula** is a formula that is in **disjunctive normal form**: it is an OR of zero or more **clauses**, each of which is an AND of variables or their negations. An example would be $(x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge x_4) \vee x_2)$. The **#DNF** problem is to count the number of satisfying assignments of a formula presented in disjunctive normal form.

Solving #DNF exactly is #**P**-complete, so we don't expect to be able to do it. Instead, we'll get a FPRAS by cleverly sampling solutions. The need for cleverness arises because just sampling solutions directly by generating one of the 2^n possible assignments to the n variables may find no satisfying assignments at all.

The trick is to sample pairs (x, i), where x assignment that satisfies clause C_i. For each pair (x, i), define $f(x, i) = 1$ if and only if $C_j(x) = 0$ for all $j < i$; then $\sum_{(x,i)} f(x, i)$ counts every satisfying assignment exactly once. It is also the case that if we can sample the (x, i) uniformly, the proportion ρ of good pairs is at least $1/m$, since every satisfying assignment x contributes at most m pairs total, and one of them is good.

The only tricky part is figuring out how to sample pairs (x, i) with $C_i(x) = 1$ so that all pairs occur with the same probability. Let $U_i = \{(x, i) | C_i(x) = 1\}$. Then we can compute $|U_i| = 2^{n-k_i}$ where k_i is the number of literals in C_i. Using this information, we can sample i first with probability $|U_i| \,/\, \sum_j |U_j|$, then sample x from U_i just by picking values for the $n - k$ variables not fixed by C_i.

With $N \geq \frac{4}{\epsilon^2(1/m)} \ln \frac{2}{\delta} = \frac{4m}{\epsilon^2} \ln \frac{2}{\delta}$, we obtain an estimate $\hat{\rho}$ for the proportion of pairs (x, i) with $f(x, i) = 1$ that is within ϵ relative error of ρ with probability at least $1 - \delta$. Multiplying this by $\sum |U_i|$ then gives the desired count of satisfying assignments.

8.4 Approximating #KNAPSACK

Here is an algorithm for approximating the number of solutions to a **KNAPSACK** problem, due to Dyer [Dye03]. We'll concentrate on the simplest version, 0–1 KNAPSACK, following the analysis in Section 2.1 of [Dye03].

For the 0–1 KNAPSACK problem, we are given a set of n objects of weight $0 \leq a_1 \leq a_2 \leq \ldots a_n \leq b$, and we want to find a 0–1 assignment $x_1, x_2, \ldots, x_n$ such that $\sum_{i=1}^{n} a_i x_i \leq b$ (usually while optimizing some property that prevents us from setting all the x_i to zero). We'll assume that the a_i and b are all integers.

For #KNAPSACK, we want to compute $|S|$, where S is the set of all assignments to the x_i that make $\sum_{i=1}^{n} a_i x_i \leq b$.

There is a well-known **fully polynomial-time approximation scheme** for optimizing KNAPSACK, based on dynamic programming. The idea is that a maximum-weight solution can be found exactly in time polyomial in b, and if b is too large, we can reduce it by rescaling all the a_i and b at the cost of a small amount of error. A similar idea is used in Dyer's algorithm: the KNAPSACK problem is rescaled so that size of the solution set S' of the rescaled version can be computed in polynomial time. Sampling is then used to determine what proportion of the solutions in S' correspond to solutions of the original problem.

Scaling step: Let $a'_i = \lfloor n^2 a_i / b \rfloor$. Then $0 \leq a'_i \leq n^2$ for all i. Taking the floor creates some error: if we try to reconstruct a_i from a'_i, the best we can do is argue that $a'_i \leq n^2 a_i / b < a'_i + 1$ implies $(b/n^2)a'_i \leq a_i < (b/n^2)a'_i + (b/n^2)$. The reason for using n^2 as our rescaled bound is that the total error in the upper bound on a_i, summed over all i, is bounded by $n(b/n^2) = b/n$, a fact that will become important soon.

Let $S' = \{\vec{x} | \sum_{i=1}^{n} a'_i x_i \leq n^2\}$ be the set of solutions to the rescaled knapsack problem, where we substitute a'_i for a_i and $n^2 = (n^2/b)b$ for b. Claim: $S \subseteq S'$. Proof: $\vec{x} \in S$ if and only if $\sum_{i=1}^{n} a_i x_i \leq b$. But then

$$\begin{aligned}
\sum_{i=1}^{n} a'_i x_i &= \sum_{i=1}^{n} \lfloor n^2 a_i / b \rfloor \, x_i \\
&\leq \sum_{i=1}^{n} (n^2/b) a_i x_i \\
&= (n^2/b) \sum_{i=1}^{n} a_i x_i \\
&\leq n^2,
\end{aligned}$$

which shows $\vec{x} \in S'$.

The converse does not hold. However, we can argue that any $\vec{x} \in S'$ can be shoehorned into S by setting at most one of the x_i to 0. Consider the set of all positions i such that $x_i = 1$ and $a_i > b/n$. If this set is empty, then $\sum_{i=1}^{n} a_i x_i \leq \sum_{i=1}^{n} b/n = b$, and $\vec{x}$ is already in S. Otherwise, pick any position i with $x_i = 1$ and $a_i > b/n$, and let $y_j = 0$ when $j = i$ and $y_j = x_j$ otherwise. Then

$$\begin{aligned}
\sum_{j=1}^{n} a_j y_j &= \sum_{j=1}^{n} a_j x_j - a_i \\
&< \sum_{j=1}^{n} ((b/n^2)a'_j + b/n^2)x_j - b/n \\
&\leq (b/n^2) \sum_{j=1}^{n} a'_j x_j + b/n - b/n \\
&\leq (b/n^2)n^2 \\
&= b.
\end{aligned}$$

Applying this mapping to all elements $\vec{x}$ of S maps at most $n+1$ of them to each $\vec{y}$ in S'; it follows that $|S'| \leq (n+1)\,|S|$, which means that if we can sample elements of S' uniformly, each sample will hit S with probability at least $1/(n+1)$.

To compute $|S'|$, let $C(k,m) = \left|\{\vec{x} | \sum_{i=1}^{k} a'_i x_i \leq m\}\right|$ be the number of subsets of $\{a'_1, \ldots, a'_k\}$ that sum to m or less. Then $C(k,m)$ satisfies the recurrence

$$\begin{aligned}
C(k,m) &= C(k-1, m-a'_k) + C(k-1,m) \\
C(0,m) &= 1
\end{aligned}$$

where k ranges from 0 to n and m ranges from 0 to n^2, and we treat $C(k-1, m-a'_k) = 0$ if $m - a'_k < 0$. The idea is that $C(k-1, m-a'_k)$ counts all the ways to make m if we include a'_k, and $C(k-1,m)$ counts all the ways to make m if we exclude it. The base case corresponds to the empty set (which sums to $\leq m$ no matter what m is).

We can compute a table of all values of $C(k,m)$ by iterating through m in increasing order; this takes $O(n^3)$ time. At the end of this process, we can read off $|S'| = C(n, n^2)$.

But we can do more than this: we can also use the table of counts to sample uniformly from S'. The probability that $x'_n = 1$ for a uniform random

element of S' is exactly $C(n-1, n^2 - a'_n)/C(n, n^2)$; having chosen $x'_n = 1$ (say), the probability that $x'_{n-1} = 1$ is then $C(n-2, n^2 - a'_n - a'_{n-1})/C(n-2, n^2 - a'_n)$, and so on. So after making $O(n)$ random choices (with $O(1)$ arithmetic operations for each choice to compute the probabilities) we get a uniform element of S', which we can test for membership in S in an additional $O(n)$ operations.

We've already established that $|S| / |S'| \geq 1/(n+1)$, so we can apply Lemma 8.2.1 to get ϵ relative error with probability at least $1 - \delta$ using $\frac{3(n+1)}{\epsilon^2} \ln \frac{2}{\delta}$ samples. This gives a cost of $O(n^2 \log(1/\delta)/\epsilon^2)$ for the sampling step, or a total cost of $O(n^3 + n^2 \log(1/\delta)/\epsilon^2)$ after including the cost of building the table (in practice, the second term will dominate unless we are willing to accept $\epsilon = \omega(1/\sqrt{n})$).

It is possible to improve this bound. Dyer [Dye03] shows that using randomized rounding on the a'_i instead of just truncating them gives a FPRAS that runs in $O(n^{5/2}\sqrt{\log(1/\epsilon)} + n^2/\epsilon^2)$ time.

Chapter 9

Hashing

These are theoretical notes on hashing based largely on [MR95, §§8.4-8.5] (which is in turn based on work of Carter and Wegman [CW77] on universal hashing and Fredman, Komlós, and Szemerédi [FKS84] on $O(1)$ worst-case hashing); on [PR04] and [Pag06] for cuckoo hashing; and [MU05, §5.5.3] for Bloom filters.

9.1 Hash tables

Here we review the basic idea of **hash tables**, which are implementations of the **dictionary** data type mapping keys to values. The basic idea of hash tables is usually attributed to Dumey [Dum56].[1]

Suppose we want to store n elements from a universe U of in a table with **keys** or **indices** drawn from an index space M of size m. Typically we assume $U = [|U|] = \{0 \dots |U| - 1\}$ and $M = [m] = \{0 \dots m - 1\}$.

If $|U| \leq m$, we can just use an array. Otherwise, we can map keys to positions in the array using a **hash function** $h : U \to M$. This necessarily produces **collisions**: pairs (x, y) with $h(x) = h(y)$, and any design of a hash table must include some mechanism for handling keys that hash to the same place. Typically this is a secondary data structure in each bin, but we may also place excess values in some other place. Typical choices for data structures are linked lists (**separate chaining** or just **chaining**) or secondary hash tables (see Section 9.3 below). Alternatively, we can push

[1] Caveat: This article is pretty hard to find, so I am basing this citation on its frequent appearance in later sources. This is generally a bad idea that is only barely justified by the fact that lecture notes are sometimes not held to the same level of scholarship as actual publications.

U	Universe of all keys
$S \subseteq U$	Set of keys stored in the table
$n = \|S\|$	Number of keys stored in the table
M	Set of table positions
$m = \|M\|$	Number of table positions
$\alpha = n/m$	Load factor

Table 9.1: Hash table parameters

excess values into other positions in the same hash table (**open addressing** or **probing**) or another hash table (see Section 9.4).

For all of these techniques, the cost will depend on how likely it is that we get collisions. An adversary that knows our hash function can always choose keys with the same hash value, but we can avoid that by choosing our hash function randomly. Our ultimate goal is to do each search in $O(1 + n/m)$ expected time, which for $n \leq m$ will be much better than the $\Theta(\log n)$ time for pointer-based data structures like balanced trees or skip lists. The quantity n/m is called the **load factor** of the hash table and is often abbreviated as α.

All of this only works if we are working in a RAM (random-access machine model), where we can access arbitrary memory locations in time $O(1)$ and similarly compute arithmetic operations on $O(\log |U|)$-bit values in time $O(1)$. There is an argument that in reality any actual RAM machine requires either $\Omega(\log m)$ time to read one of m memory locations (routing costs) or, if one is particularly pedantic, $\Omega(m^{1/3})$ time (speed of light + finite volume for each location). We will ignore this argument.

We will try to be consistent in our use of variables to refer to the different parameters of a hash table. Table 9.1 summarizes the meaning of these variable names.

9.2 Universal hash families

A family of hash functions H is **2-universal** if for any $x \neq y$, $\Pr[h(x) = h(y)] \leq 1/m$ for a uniform random $h \in H$. It's **strongly 2-universal** if for any $x_1 \neq x_2 \in U$, $y_1, y_2 \in M$, $\Pr[h(x_1) = y_1 \wedge h(x_2) = y_2] \leq 1/m^2$ for a uniform random $h \in H$.

For $k > 2$, **k-universal** usually means **strongly k-universal**: Given distinct $x_1 \ldots x_k$, and any $y_1 \ldots y_k$, $\Pr[h(x_i) = y_i) \forall i] \leq m^{-k}$. It is possible to generalize the weak version of 2-universality to get a weak version of

k-universality ($\Pr[h(x_i)\text{are all equal}] \leq m^{-(k-1)}$), but this generalization is not as useful as strong k-universality.

To analyze universal hash families, it is helpful to have some notation for counting collisions. Let $\delta(x, y, h) = 1$ if $x \neq y$ and $h(x) = h(y)$, 0 otherwise. Abusing notation, we also define, for sets X, Y, and H, $\delta(X, Y, H) = \sum_{x \in X, y \in Y, h \in H} \delta(x, y, h)$, with e.g. $\delta(x, Y, h) = \delta(\{x\}, Y, \{h\})$. Now the statement that H is 2-universal becomes $\forall x, y : \delta(x, y, H) \leq |H|/m$; this says that only a fraction of $1/m$ of the functions in H cause any particular distinct x and y to collide.

If H includes all functions $U \to M$, we get equality: a random function gives $h(x) = h(y)$ with probability exactly $1/m$. But we might do better if each h tends to map distinct values to distinct places. The following lemma shows we can't do too much better:

Lemma 9.2.1. *For any family H, there exist x, y such that $\delta(x, y, H) \geq \frac{|H|}{m}\left(1 - \frac{m-1}{|U|-1}\right)$.*

Proof. We'll count collisions in the inverse image of each element z. Since all distinct pairs of elements of $h^{-1}(z)$ collide with each other, we have

$$\delta(h^{-1}(z), h^{-1}(z), h) = \left|h^{-1}(z)\right| \cdot \left(\left|h^{-1}(z)\right| - 1\right).$$

Summing over all $z \in M$ gets all collisions, giving

$$\delta(U, U, h) = \sum_{z \in M} \left(\left|h^{-1}(z)\right| \cdot \left(\left|h^{-1}(z)\right| - 1\right)\right).$$

Use convexity or Lagrange multipliers to argue that the right-hand side is minimized subject to $\sum_z \left|h^{-1}(z)\right| = |U|$ when all pre-images are the same size $|U|/m$. It follows that

$$\begin{aligned}
\delta(U, U, h) &\geq \sum_{z \in M} \frac{|U|}{m}\left(\frac{|U|}{m} - 1\right) \\
&= m\frac{|U|}{m}\left(\frac{|U|}{m} - 1\right) \\
&= \frac{|U|}{m}(|U| - m).
\end{aligned}$$

If we now sum over all h, we get

$$\delta(U, U, H) \geq \frac{|H|}{m}|U|\left(|U| - m\right).$$

There are exactly $|U|(|U|-1)$ ordered pairs x, y for which $\delta(x, y, H)$ might not be zero; so the Pigeonhole principle says some pair x, y has

$$\begin{aligned}\delta(x, y, H) &\geq \frac{|H|}{m}\left(\frac{|U|(|U|-m)}{|U|(|U|-1)}\right) \\ &= \frac{|H|}{m}\left(1-\frac{m-1}{|U|-1}\right).\end{aligned}$$

□

Since $1-\frac{m-1}{|U|-1}$ is likely to be very close to 1, we are happy if we get the 2-universal upper bound of $|H|/m$.

Why we care about this: With a 2-universal hash family, chaining using linked lists costs $O(1+s/n)$ expected time per operation. The reason is that the expected cost of an operation on some key x is proportional to the size of the linked list at $h(x)$ (plus $O(1)$ for the cost of hashing itself). But the expected size of this linked list is just the expected number of keys y in the dictionary that collide with x, which is exactly $s\delta(x, y, H) \leq s/n$.

9.2.1 Example of a 2-universal hash family

Universal hash families often look suspiciously like classic pseudorandom number generators. Here is a 2-universal hash family based on taking remainders.

Lemma 9.2.2. *Let $h_{ab}(x) = (ax+b \bmod p) \bmod m$, where $a \in \mathbb{Z}_p - \{0\}, b \in \mathbb{Z}_p$, and p is a prime $\geq m$. Then $\{h_{ab}\}$ is 2-universal.*

Proof. Again, we count collisions. Split $h_{ab}(x)$ as $g(f_{ab}(x))$ where $f_{ab}(x) = ax + b \bmod p$ and $g(x) = x \bmod m$.

Claim: If $x \neq y$, then $\delta(x, y, H) = \delta(\mathbb{Z}_p, \mathbb{Z}_p, g)$. Proof: Let $r \neq s \in Z_p$. Then $ax+b = r$ and $ay+b = s$ has a unique solution mod p (because $\mathbb{Z}_p$ is a finite field). This implies that for fixed x and y, $(f_{ab}(x), f_{ab}(y))$ is a bijection between pairs a, b and pairs r, s and thus $\delta(x, y, H) = \sum_{r \neq s} \delta(r, s, g) = \delta(\mathbb{Z}_p, \mathbb{Z}_p, g)$.

For fixed r, $\lceil p/m \rceil$ is an upper bound on the number of elements in $g^{-1}(g(r))$; subtract one to get the number of s that collide with r. So the total number of collisions $\delta(\mathbb{Z}_p, \mathbb{Z}_p, g) \leq p(\lceil p/m \rceil - 1) \leq p(p-1)/m = |H|/m$. □

A difficulty with this hash family is that it requires doing modular arithmetic. A faster hash is given by Dietzfelbinger *et al.* [DHKP97], although it

requires a slight weakening of the notion of 2-universality. For each k and ℓ they define a class $H_{k,\ell}$ of functions from $[2^k]$ to $[2^\ell]$ by defining

$$h_a(x) = (ax \bmod 2^k) \operatorname{div} 2^{k-\ell},$$

where $x \operatorname{div} y = \lfloor x/y \rfloor$. They prove [DHKP97, Lemma 2.4] that if a is a random *odd* integer with $0 < a < 2^\ell$, and $x \neq y$, $\Pr[h_a(x) = h_a(y)] \leq 2^{-\ell+1}$. This increases by a factor of 2 the likelihood of a collision, but any extra costs from this can often be justified in practice by the reduction in costs from working with powers of 2.

9.3 FKS hashing

Goal is to hash a static set S so that we never pay more than constant time for search (not just in expectation), while at the same time not consuming too much space.

If we are lucky in our choice of S, we may be able to do this just by hashing. A **perfect hash function** for a set $S \subseteq U$ is a hash function $h : U \to M$ that is injective on S (that is, $x \neq y \Rightarrow h(x) \neq h(y)$ when $x, y \in S$). Unfortunately, we can only count on finding a perfect hash function if m is large:

Lemma 9.3.1. *If H is 2-universal and $|S| = n$ with $n^2 \leq m$, then there is a perfect $h \in H$ for S.*

Proof. We'll do the usual collision-counting argument. For all $x \neq y$, we have $\delta(x, y, H) \leq |H|/m$. So $\delta(S, S, H) \leq n(n-1)|H|/m$. Pigeonhole principle says that there exists a particular $h \in H$ with $\delta(S, S, h) \leq n(n-1)/m < n^2/m \leq 1$. But $\delta(S, S, h)$ is an integer, so it can only be less than 1 by being equal to 0: no collisions. □

Essentially the same argument shows that if $n^2 \leq \alpha m$, then $\Pr[h \text{ is perfect for } S] \geq 1 - \alpha$. This can be handy if we want to find a perfect hash function and not just demonstrate that it exists.

Using a perfect hash function, we get $O(1)$ search time using $O(n^2)$ space. But we can do better by using perfect hash functions only at the second level of our data structure, which at top level will just be an ordinary hash table. This is the idea behind the Fredman-Komlós-Szemerédi (FKS) hash table [FKS84].

The short version is that we hash to $n = |S|$ bins, then rehash perfectly within each bin. The top-level hash table stores a pointer to a header for

each bin, which gives the size of the bin and the hash function used within it. The i-th bin, containing n_i elements, uses $O(n_i^2)$ space to allow perfect hashing. The total size is $O(n)$ as long as we can show that $\sum_{i=1}^{n} n_i^2 = O(n)$. The time to do a search is $O(1)$ in the worst case: $O(1)$ for the outer hash plus $O(1)$ for the inner hash.

Theorem 9.3.2. *The FKS hash table uses $O(n)$ space.*

Proof. Suppose we choose $h \in H$ as the outer hash function, where H is some 2-universal family of hash functions. Compute:

$$\begin{aligned}\sum_{i=1}^{n} n_i^2 &= \sum_{i=1}^{n} (n_i + n_i(n_i - 1)) \\ &= n + \delta(S, S, h).\end{aligned}$$

Since H is 2-universal, we have $\delta(S, S, H) \leq |H|\, s(s-1)/n$. But then the Pigeonhole principle says there exists some $h \in H$ with $\delta(S, S, h) \leq \frac{1}{|H|}\delta(S, S, H) \leq n(n-1)/n = n - 1$. This gives $\sum_{i=1}^{n} n_i^2 \leq n + (n-1) = 2n - 1 = O(n)$. □

If we want to find a good h quickly, increasing the size of the outer table to n/α gives us a probability of at least $1 - \alpha$ of getting a good one, using essentially the same argument as for perfect hash functions.

9.4 Cuckoo hashing

Goal: Get $O(1)$ search time in a dynamic hash table at the cost of a messy insertion procedure. In fact, each search takes only two reads, which can be done in parallel; this is optimal by a lower bound of Pagh [Pag01], which shows a matching upper bound for static dictionaries. **Cuckoo hashing** is an improved version of this result that allows for dynamic insertions.

Cuckoo hashing was invented by Pagh and Rodler [PR04]; the version described here is based on a simplified version from notes of Pagh [Pag06] (the main difference is that it uses just one table instead of the two tables—one for each hash function—in [PR04]).

9.4.1 Structure

We have a table T of size n, with two separate, independent hash functions h_1 and h_2. These functions are assumed to be k-universal for some sufficiently large value k; as long as we never look at more than k values at once, this

means we can treat them effectively as random functions. (In practice, using crummy hash functions seems to work just fine, a common property of hash tables.)

Every key x is stored either in $T[h_1(x)]$ or $T[h_2(x)]$. So the search procedure just looks at both of these locations and returns whichever one contains x (or fails if neither contains x).

To insert a value $x_1 = x$, we must put it in $T[h_1(x_1)]$ or $T[h_2(x_1)]$. If one or both of these locations is empty, we put it there. Otherwise we have to kick out some value that is in the way (this is the "cuckoo" part of cuckoo hashing, named after the bird that leaves its eggs in other birds' nests). We do this by letting $x_2 = T[h_1(x_1)]$ and writing x_1 to $T[h_1(x_1)]$. We now have a new "nestless" value x_2, which we swap with whatever is in $T[h_2(x_2)]$. If that location was empty, we are done; otherwise, we get a new value x_3 that we have to put in $T[h_1(x_3)]$ and so on. The procedure terminates when we find an empty spot or if enough iterations have passed that we don't expect to find an empty spot, in which case we rehash the entire table. This process can be implemented succinctly as shown in Algorithm 9.1.

A detail not included in the above code is that we always rehash (in theory) after m^2 insertions; this avoids potential problems with the hash functions used in the paper not being universal enough. We will avoid this issue by assuming that our hash functions are actually random (instead of being approximately n-universal with reasonably high probability). For a more principled analysis of where the hash functions come from, see [PR04].

9.4.2 Analysis

The main question is how long it takes the insertion procedure to terminate, assuming the table is not too full.

First let's look at what happens during an insert if we have a lot of nestless values. We have a sequence of values $x_1, x_2, \ldots$, where each pair of values x_i, x_{i+1} collides in h_1 or h_2. Assuming we don't reach the loop limit, there are three main possibilities (the leaves of the tree below):

1. Eventually we reach an empty position without seeing the same key twice.

2. Eventually we see the same key twice; there is some i and $j > i$ such that $x_j = x_i$. Since x_i was already moved once, when we reach it the second time we will try to move it back, displacing x_{i-1}. This process continues until we have restored x_2 to $T[h_1(x_1)]$, displacing x_1

```
1  procedure insert(x)
2  begin
3      if T(h_1(x)) = x or T(h_2(x)) = x then
4          return
5      end if
6      pos ← h_1(x)
7      for i ← 1 ... n do
8          if T[pos] = ⊥ then
9              T[pos] ← x
10             return
11         end if
12         x ⇄ T[pos]
13         if pos = h_1(x) then
14             pos ← h_2(x)
15         else
16             pos ← h_1(x)
17         end if
18     end for
19     If we got here, rehash the table and reinsert x.
20 end
```

Algorithm 9.1: Insertion procedure for cuckoo hashing. Adapted from [Pag06].

to $T[h_2(x_1)]$ and possibly creating a new sequence of nestless values. Two outcomes are now possible:

(a) Some x_l is moved to an empty location. We win!

(b) Some x_l is moved to a location we've already looked at. We lose! In this case we are playing musical chairs with more players than chairs, and have to rehash.

Let's look at the probability that we get the last, *closed loop* case. Following Pagh-Rodler, we let v be the number of distinct nestless keys in the loop. We can now count how many different ways such a loop can form: There are v^3 choices for i, j, and l, m^{v-1} choices of cells for the loop, and n^{v-1} choices for the non-x_1 elements of the loop. We also have $2v$ edges each of which occurs with probability m^{-1}, giving a total probability of $v^3 m^{v-1} n^{v-1} m^{-2v} = v^3 (n/m)^v / nm$. Summing this over all v gives $\frac{1}{nm} \sum v^3 (n/m)^v = O\left(\frac{1}{nm}\right)$ (the series converges under the assumption that $n/m < 1$). Since the cost of hitting a closed loop is $O(n+m)$, this adds $O(1)$ to the insertion complexity.

Now we look at what happens if we don't get a closed loop. It's a little messy to analyze the behavior of keys that appear more than once in the sequence, so the trick used in the paper is to observe that for any sequences of nestless keys $x_1 \ldots x_p$, there is a subsequence of size $p/3$ with no repetitions that starts with x_1. Since there are only two subsequences that start with x_1 (we can't have the same key show up more than twice), this will either be $x_1 \ldots x_{j-1}$ or $x_1 = x_{i+j-1} \ldots x_p$, and a case analysis shows that at least one of these will be big. We can then argue that the probability that we get a sequence of v distinct keys starting with x_1 in T is at most $2(n/m)^{v-1}$ (since we have to hit a nonempty spot, with probability at most n/m, at each step, but there are two possible starting locations), which gives an expected insertion time bounded by $\sum 3v(n/m)^{v-1} = O(1)$.

9.4.3 Practical issues

For large hash tables, local probing schemes are faster, because it is likely that all of the locations probed to find a particular value will be on the same virtual memory page. This means that a search for a new value usually requires one cache miss instead of two. **Hopscotch hashing** [HST08] combines ideas from linear probing and cuckoo hashing to get better performance than both in practice.

Hash tables that depend on strong properties of the hash function may behave badly if the user supplies a crummy hash function. For this reason,

many library implementations of hash tables are written defensively, using algorithms that respond better in bad cases. See `http://svn.python.org/view/python/trunk/Objects/dictobject.c` for an example of a widely-used hash table implementation chosen specifically because of its poor theoretical characteristics.

9.5 Bloom filters

See [MU05, §5.5.3] for basics and a formal analysis or `http://en.wikipedia.org/wiki/Bloom_filter` for many variations and the collective wisdom of the unwashed masses. The presentation here mostly (in some cases, slavishly) follows [MU05].

Bloom filters are a highly space-efficient randomized data structure invented by Burton H. Bloom [Blo70] that store sets of data, with a small probability that elements not in the set will be erroneously reported as being in the set.

Suppose we have k independent hash functions $h_1, h_2, \ldots, h_k$. Our memory store A is a vector of m bits, all initially zero. To store a key x, set $A[h_i(x)] = 1$ for all i. To test membership for x, see if $A[h_i(x)] = 1$ for all i. The membership test always gives the right answer if x is in fact in the Bloom filter. If not, we might decide that x is in the Bloom filter anyway.

9.5.1 False positives

The probability of such **false positives** can be computed in two steps: first, we estimate how many of the bits in the Bloom filter are set after inserting n values, and then we use this estimate to compute a probability that any fixed x shows up when it shouldn't.

If the h_i are close to being independent random functions,[2] then with n entries in the filter we have $\Pr[A[i] = 1] = 1 - (1 - 1/m)^{kn} \leq 1 - e^{-kn/m} = 1 - e^{-k\alpha}$, where $\alpha = n/m$ is the load factor, since each of the kn bits that we set while inserting the n values has one chance in m of hitting position i.

In aggregate, this gives us an expected $\mu = m(1 - (1 - 1/m)^{kn}) \leq m(1 - e^{-k\alpha})$ one bits in the array. If we had exactly this number, the probability of a false positive would at most $(1 - e^{-k\alpha})^k$, since each $A[h_i(x)]$

[2] We are going sidestep the rather deep swamp of how plausible this assumption is and what assumption we should be making instead; however, it is known [KM08] that starting with two sufficiently random-looking hash functions h and h' and setting $h_i(x) = h(x) + ih'(x)$ works.

would hit a one with independent probability at most $1-e^{-k\alpha}$. The problem is that these probabilities are not independent: if $A[h_1(x)]$ is more likely than we expect to be a one (because more than the expected number of bits are one), so is $A[h_2(x)]$ and so forth.

But this effect is not large. Applying Chernoff bounds (e.g., (3.2.3)), we have

$$\Pr[\sum_{i=1}^{m} A[i] \geq (1+\delta)m(1-e^{-k\alpha})] \leq e^{-\mu\delta^2/3},$$

which is bounded by n^{-c} for $\delta \geq 3c\sqrt{\frac{\log n}{\mu}}$. So as long as a reasonably large fraction of the array is likely to be full, the relative error on the $(1-e^{-k\alpha})^k$ approximation is going to be $O\left(k\sqrt{\frac{\log n}{\mu}}\right)$—a tiny difference when μ is large. If μ is small (particular if it is small relative to m), then we don't care about the relative error so much because the probability of getting a false positive will already be exponentially small as a function of k.

So let's assume for simplicity that our false positive probability is exactly $(1-e^{-k\alpha})^k$. We can choose k to minimize this quantity for fixed α by doing the usual trick of taking the derivative with respect to k and setting it to zero; to avoid weirdness with the k in the exponent, it helps to take the logarithm first (which doesn't affect the location of the minimum). I know of no principled way to solve the resulting equation,[3] but plugging in $k = \frac{1}{\alpha}\ln 2$ miraculously works:

$$\begin{aligned}
\frac{d}{dk}\ln(1-e^{-k\alpha})^k &= \frac{d}{dk}\left(k\ln(1-e^{-k\alpha})\right) \\
&= \ln(1-e^{-k\alpha}) + \frac{k\alpha\cdot e^{-k\alpha}}{1-e^{-k\alpha}} \\
&= \ln(1-e^{-\ln 2}) + \frac{\ln 2\cdot e^{-\ln 2}}{1-e^{-\ln 2}} \\
&= \ln\left(1-\frac{1}{2}\right) + \frac{\frac{1}{2}\ln 2}{1-\frac{1}{2}} \\
&= -\ln 2 + \ln 2 \\
&= 0.
\end{aligned}$$

[3] A student who wished to remain anonymous suggested substituting x for $e^{-k\alpha}$, which gives (after taking the derivative of the logarithm) $\ln(1-x) - \frac{x\ln x}{1-x} = 0$ or $x\ln x = (1-x)\ln(1-x)$. Symmetry gives a solution at $x = 1-x = 1/2$, giving $k = \frac{1}{\alpha}\ln 2$ as claimed.

In other words, to minimize the false positive rate for a known load factor α, we want to choose $k = \frac{1}{\alpha} \ln 2$, which makes each bit one with probability $1 - e^{-\ln 2} = \frac{1}{2}$. This makes intuitive sense, since having each bit be one or zero with equal probability maximizes the entropy of the data.

The probability of a false positive is then $2^{-k} = 2^{-\ln 2/\alpha}$. For a given maximum false positive rate ϵ, and assuming optimal choice of k, we need to keep $\alpha \leq \frac{\ln^2 2}{\ln(1/\epsilon)}$.

Alternatively, if we fix ϵ and n, we need $m \geq n \cdot \frac{\ln(1/\epsilon)}{\ln^2 2} \approx 1.442 \lg(1/\epsilon)$. This is very good for constant ϵ.

9.5.2 Applications

Bloom filters are popular in networking and database systems because they can be used as a cheap test to see if some key is actually present in a data structure that it's otherwise expensive to search in. Bloom filters are particular nice in hardware implementations, since the k hash functions can be computed in parallel.

An example is the **Bloomjoin** in distributed databases [ML86]. Here we want to do a join on two tables stored on different machines (a join is an operation where we find all pairs of rows, one in each table, that match on some common key). A straightforward but expensive way to do this is to send the list of keys from the smaller table across the network, then match them against the corresponding keys from the larger table. If there are n_s rows in the smaller table, n_b rows in the larger table, and j matching rows in the larger table, this requires sending n keys plus j rows. If instead we send a Bloom filter representing the set of keys in the smaller table, we only need to send $\lg(1/\epsilon)/\ln 2$ bits for the Bloom filter plus and extra ϵn_b rows on average for the false positives. This can be cheaper than sending full keys across if the number of false positives is reasonably small.

9.5.3 Comparison to optimal space

If we wanted to design a Bloom-filter-like data structure from scratch and had no constraints on processing power, we'd be looking for something that stored an index of size $\lg M$ into a family of subsets $S_1, S_2, \ldots S_M$ of our universe of keys U, where $|S_i| \leq \epsilon |U|$ for each i (giving the upper bound on the false positive rate) and for any set $A \subseteq U$ of size n, $A \subseteq S_i$ for at least one S_i (allowing us to store A).

Let $N = |U|$. Then each set S_i covers $\binom{\epsilon N}{n}$ of the $\binom{N}{n}$ subsets of size n. If we could get them to overlap optimally (we can't), we'd still need a minimum

of $\binom{N}{n} \Big/ \binom{\epsilon N}{n} = (N)_n/(\epsilon N)_n \approx (1/\epsilon)^n$ sets to cover everybody, where the approximation assumes $N \gg n$. Taking the log gives $\lg M \approx n \lg(1/\epsilon)$, meaning we need about $\lg(1/\epsilon)$ bits per key for the data structure. Bloom filters use $1/\ln 2$ times this.

There are known data structures that approach this bound asymptotically; see Pagh *et al.* [PPR05]. These also have other desirable properties, like supporting deletions and faster lookups if we can't look up bits in parallel. As far as I know, they are not used much in practice.

9.5.4 Counting Bloom filters

It's not hard to modify a Bloom filter to support deletion. The basic trick is to replace each bit with a counter, so that whenever a value x is inserted, we increment $A[h_i(x)]$ for all i and when it is deleted, we decrement the same locations. The search procedure now returns $\min_i A[h_i(x)]$ (which means that it principle it can even report back multiplicities, though with some probability of reporting a value that is too high). To avoid too much space overhead, each array location is capped at some small maximum value c; once it reaches this value, further increments have no effect. The resulting structure is called a **counting Bloom filter**, due to Fan *et al.* [FCAB00].

Fan *et al.* observe that the probability that the m table entries include one that is at least c after n insertions is bounded by $m\binom{nk}{c}\frac{1}{m^c} \leq m\left(\frac{enk}{cm}\right)^c = m(ek\alpha/c)^c$. For $k = \frac{1}{\alpha}\ln 2$, this is $m(e \ln 2/c)^c$. For the specific value of $c = 16$ (corresponding to 4 bits per entry), they compute a bound of $1.37 \times 10^{-15}m$, which they argue is minuscule for all reasonable values of m (it's a systems paper).

The possibility that a long chain of alternating insertions and deletions might produce a false negative due to overflow is considered in the paper, but "the probability of such a chain of events is so low that it is much more likely that the proxy server would be rebooted in the meantime and the entire structure reconstructed." An alternative way of dealing with this problem is to never decrement a maxed-out register; this never produces a false negative, but may cause the filter to slowly fill up with maxed-out registers, producing a higher false-positive rate.

A fancier variant of this idea is the **spectral Bloom filter** [CM03], which uses larger counters to track multiplicities of items (essentially, we can guess that the number of times a particular value x was inserted is equal to $\min_{i=1}^{m} A[h_i(x)]$), with some extra tinkering to detect errors based on deviations from the typical joint distribution of the $A[h_i(x)]$ values.

9.5.5 Count-min sketches

An idea similar to counting Bloom filters is used in **data stream computation**. In this model, we are given a huge flood of data—far too big to store—in a single pass, and want to incrementally build a small data structure, called a **sketch**, that will allow us to answer statistical questions about the data after we've processed it all. The motivation is the existence of data sets that are too large to store at all (e.g., network traffic statistics), or too large to store in fast memory (e.g., very large database tables). By building a sketch we can make one pass through the data set but answer queries after the fact, with some loss of accuracy.

An example of a problem in this model is that we are presented with a sequence of pairs (i_t, c_t) where $1 \le i_t \le n$ is an *index* and c_t is a *count*, and we want to concstruct a sketch that will allows us to approximately answer statistical queries about the vector a given by $a_i = \sum_{t,i[t]=i} c_t$. The size of the sketch should be polylogarithmic in the size of a and the length of the stream, and polynomial in the error bounds. Updating the sketch given a new data point should be cheap.

A solution to this problem is given by the **count-min sketch** of Cormode and Muthukrishnan [CM05] (see also [MU05, §13.4]). This gives approximations of a_i, $\sum_{i=\ell}^{r} a_i$, and $a \cdot b$ (for any fixed b), and can be used for more complex tasks like finding **heavy hitters**—indices with high weight. The easiest case is approximating a_i when all the c_t are non-negative, so we'll start with that.

9.5.5.1 Initialization and updates

To construct a count-min sketch, build a two-dimensional array c with width $w = \lceil e/\epsilon \rceil$ and depth $d = \lceil \ln(1/\delta) \rceil$, where ϵ is the error bound and δ is the probability of exceeding the error bound. Choose d independent hash functions from some 2-universal hash family. Initialize c to all zeros.

The update rule: Given an update (i_t, c_t), increment $c[j, h_j(i_t)]$ by c_t for $j = 1 \ldots d$. (This is the *count* part of count-min.)

9.5.5.2 Queries

Let's start with **point queries**. Here we want to estimate a_i for some fixed i. There are two cases, depending on whether the increments are all non-negative, or arbitrary. In both cases we will get an estimate whose error is linear in both the error parameter ϵ and the ℓ_1-norm $\|a\|_1 = \sum_i |a_i|$ of a. It follows that the relative error will be low for heavy points, but we may

get a large relative error for light points (and especially large for points that don't appear in the data set at all).

For the non-negative case, to estimate a_i, compute $\hat{a}_i = \min_j c[j, h_j(i)]$. (This is the *min* part of coin-min.) Then:

Lemma 9.5.1. *When all c_t are non-negative, for $\hat{a}_i$ as defined above:*

$$\hat{a}_i \geq a_i, \tag{9.5.1}$$

and

$$\Pr[\hat{a}_i \leq a_i + \epsilon \|a\|_1] \geq 1 - \delta. \tag{9.5.2}$$

Proof. The lower bound is easy. Since for each pair (i, c_t) we increment each $c[j, h_j(i)]$ by c_t, we have an invariant that $a_i \leq c[j, h_j(i)]$ for all j throughout the computation, which gives $a_i \leq \hat{a}_i = \min_j c[j, h_j(i)]$.

For the upper bound, let I_{ijk} be the indicator for the event that $(i \neq k) \wedge (h_j(i) = h_j(k))$, i.e., that we get a collision between i and k using h_j. The 2-universality property of the h_j gives $\mathrm{E}[I_{ijk}] = 1/w \leq \epsilon/e$.

Now let $X_{ij} = \sum_{k=1}^{n} I_{ijk} a_k$. Then $c[j, h_j(i)] = a_i + X_{ij}$. (The fact that $X_{ij} \geq 0$ gives an alternate proof of the lower bound.) Now use linearity of expectation to get

$$\begin{aligned}
\mathrm{E}[X_{ij}] &= \mathrm{E}[\sum_{k=1}^{n} I_{ijk} a_k] \\
&= \sum_{k=1}^{n} a_k \mathrm{E}[I_{ijk}] \\
&\leq \sum_{k=1}^{n} a_k (\epsilon/e) \\
&= (\epsilon/e) \|a\|_1 .
\end{aligned}$$

So $\Pr[c[j, h_j(i)] > a_i + \epsilon \|a\|_1] = \Pr[X_{ij} > e \mathrm{E}[X_{ij}]] < 1/e$, by Markov's inequality. With d choices for j, and each h_j chosen independently, the probability that every count is too big is at most $(1/e)^{-d} = e^{-d} \leq \exp(-\ln(1/\delta)) = \delta$. □

Now let's consider the general case, where the increments c_t might be negative. We still initialize and update the data structure as described in Section 9.5.5.1, but now when computing $\hat{a}_i$, we use the median count instead of the minimum count: $\hat{a}_i = \mathrm{median}_j c[j, h_j(i)]$. Now we get:

Lemma 9.5.2. *For $\hat{a}_i$ as defined above,*

$$\Pr[a_i - 3\epsilon \|a\|_1 \le \hat{a}_i \le a_i + 3\epsilon \|a\|_1] > 1 - \delta^{1/4}. \tag{9.5.3}$$

Proof. We again define the error term X_{ij} as above, and observe that

$$\begin{aligned}
\mathrm{E}[|X_{ij}|] &= \mathrm{E}[\Big|\sum\nolimits_k I_{ijk} a_k\Big|] \\
&\le \sum_{k=1}^{n} |a_k \,\mathrm{E}_{ijk}| \\
&\le \sum_{k=1}^{n} |a_k| \,(\epsilon/e) \\
&= (\epsilon/e) \|a\|_1 .
\end{aligned}$$

Using Markov's inequality, we get $\Pr[|X_{ij}|] > 3\epsilon \|a\|_1] = \Pr[|X_{ij}| > 3e \,\mathrm{E}[X_{ij}]] < 1/3e < 1/8$. In order for the median to be off by more than $3\epsilon \|a\|_1$, we need $d/2$ of these low-probability events to occur. The expected number that occur is $\mu = d/8$, so applying the Chernoff bound we are looking at

$$\begin{aligned}
\Pr[S \ge (1+3)\mu] &\le (e^3/4^4)^{d/8} \\
&\le (e^{3/8}/2)^{\ln(1/\delta)} \\
&= \delta^{\ln 2 - 3/8} \\
&< \delta^{1/4}
\end{aligned}$$

(the actual exponent is about 0.31, but 1/4 is easier to deal with). This immediately gives (9.5.3). □

One way to think about this is that getting an estimate within $\epsilon \|a\|_1$ of the right value with probability at least $1 - \delta$ requires 3 times the width and 4 times the depth—or 12 times the space and 4 times the time—when we aren't assuming increments are non-negative.

Next, we consider inner products. Here we want to estimate $a \cdot b$, where a and b are both stored as count-min sketches using the same hash functions. The paper concentrates on the case where a and b are both non-negative, which has applications in estimating the size of a join in a database. The method is to estimate $a \cdot b$ as $\min_j \sum_{k=1}^{w} c_a[j,k] \cdot c_b[j,k]$.

For a single j, the sum consists of both good values and bad collisions; we have $\sum_{k=1}^{w} c_a[j,k] \cdot c_b[j,k] = \sum_{k=1}^{n} a_i b_i + \sum_{p \ne q,h} j_{(p)=h} j_{(q)} a_p b_q$. The second

term has expectation

$$\begin{aligned}\sum_{p\neq q} \Pr[h_j(p) = h_j(q)]a_p b_q &\leq \sum_{p\neq q} (\epsilon/e)a_p b_q \\ &\leq \sum_{p,q}, (\epsilon/e)a_p b_q \\ &\leq (\epsilon/e) \|a\|_1 \|b\|_1 .\end{aligned}$$

So as in the point-query case we get probability at most $1/e$ that a single j gives a value that's more than $\epsilon \|a\|_1 \|b\|_1$ too high, so the probability that the minimum value is too high is at most $e^{-d} \leq \delta$.

9.5.5.3 Finding heavy hitters

Here we want to find the heaviest elements in the set: those indices i for which a_i exceeds $\phi \|a\|_1$ for some constant threshold ϕ. The easy case is when increments are non-negative (for the general case, see the paper), and uses a method from a previous paper by Charikar *et al.* [CCFC04]. Instead of trying to find the elements after the fact, we extend the data structure and update procedure to track all the heavy elements found so far (stored in a heap), as well as $\|a\|_1 = \sum c_t$. When a new increment (i, c) comes in, we first update the count-min structure and then do a point query on a_i; if $\hat{a}_i \geq \phi \|a\|_1$, we insert i into the heap, and if not, we delete i along with any other value whose stored point-query estimate has dropped below threshold.

The trick here is that the threshold $\phi \|a\|_1$ only increases over time (remember that we are assuming non-negative increments). So if some element i is below threshold at time t, it can only go above threshold if it shows up again, and we have a probability of at least $1 - \delta$ of including it then.

The total space cost for this data structure is the cost of the count-min structure plus the cost of the heap; this last part will be $O((1 + \epsilon)/\phi)$ with high probability, since this is the maximum number of elements that have weight at least $\phi \|a\|_1 /(1 + \epsilon)$, the minimum needed to get an apparent weight of $\phi \|a\|_1$ even after taking into account the error in the count-min structure.

9.6 Locality-sensitive hashing

Locality-sensitive hashing was invented by Indyk and Motwani [IM98] to solve the problem of designing a data structure that finds approximate nearest neighbors to query points in high dimension. We'll mostly be following this paper in this section, concentrating on the hashing parts.

9.6.1 Approximate nearest neighbor search

In the **nearest neighbor search** problem (**NNS** for short), we are given a set of n points P in a metric space with distance function d, and we want to construct a data structure that allows us to quickly find the closet point in P to any given query point. Indyk and Motwani were particularly interested in what happens in $\mathbb{R}^d$ for high dimension d under various natural metrics. Because the volume of a ball in a high-dimensional space grows exponentially with the dimension, this problem suffers from the **curse of dimensionality**[Bel57]: simple techniques based on, for example, assigning points in P to nearby locations in a grid may require searching exponentially many grid locations. Indyk and Motwani deal with this through a combination of randomization and solving the weaker problem of **ϵ-nearest neighbor search** (ϵ-**NNS**), where it's OK to return a point p' with $d(q,p') \leq \min_{p \in P} d(q,p)$.

This problem can be solved by reduction to a simpler problem called **ϵ-point localation in equal balls** or **ϵ-PLEB**. In this problem, we are given n radius-r balls centered on points in a set C, and we want a data structure that returns a point $c' \in C$ with $d(q,c') \leq (1+\epsilon)r$ if there is at least one point c with $d(q,c) \leq r$. If there is no such point, the data structure may or may not return a point (it's allowed to say no).

The easy reduction is to use binary search. Let $R = \frac{\max_{x,y \in P} d(x,y)}{\min_{x,y \in P, x \neq y} d(x,y)}$. Given a point q, look for the minimum $\ell \in \{(1+\epsilon)^0, (1+\epsilon)^1, \ldots, R\}$ for which an ϵ-PLEB data structure with radius ℓ and centers P returns a point p with $d(q,p) \leq (1+\epsilon)\ell$; then return this point as the approximate nearest neighbor.

This requires $O(\log R)$ instances of the ϵ-PLEB data structure and $O(\log \log R)$ queries. The blowup as a function of R can be avoided using a more sophisticated data structure called a **ring-cover tree**, defined in the paper. We won't talk about ring-cover trees because they are (a) complicated and (b) not randomized. Instead, we'll move directly to the question of how we solve ϵ-PLEB.

9.6.2 Locality-sensitive hash functions

Definition 9.6.1 ([IM98])**.** *A family of hash functions H is (r_1, r_2, p_1, p_2)-**sensitive** for d if, for any points p and q, if h is chosen uniformly from H,*

1. *If $d(p,q) \leq r_1$, then $\Pr[h(p) = h(q)] \geq p_1$, and*
2. *If $d(p,q) > r_2$, then $\Pr[h(p) = h(q)] \leq p_2$.*

These are useful if $p_1 > p_2$ and $r_1 < r_2$; that is, we are more likely to hash inputs together if they are closer. Ideally, we can choose r_1 and r_2 to build ϵ-PLEB data structures for a range of radii sufficient to do binary search as described above (or build a ring-cover tree if we are doing it right). For the moment, we will aim for an (r_1, r_2)-PLEB data structure, which returns a point within r_1 with high probability if one exists, and never returns a point farther away than r_2.

9.6.2.1 Constructing an (r_1, r_2)-PLEB

The first trick is to amplify the difference between p_1 and p_2 so that we can find a point within r_1 of our query point q if one exists. This is done in three stages: First, we compute multiple hash functions to drive the probability that distant points hash together down until we get few collisions. Second, we hash our query point and target points multiple times to bring the probability that nearby points hash together up. Finally, we iterate the procedure to drive down any remaining probability of failure below a target probability δ.

Formally, for the first step, let $k = \log_{1/p_2} n$ and define a composite hash function $g(p) = (h_1(p) \dots h_k(p))$. If $d(p, q) > r_2$, $\Pr[g(p) = g(q)] \leq p_2^k = p_2^{\log_{1/p_2} n} = 1/n$. Adding this up over all n points in our data structure gives us an expected 1 false match for q.

However, we may also not be able to find the correct match for q, since p_1 may not be all that much larger than p_2. For this, we do a second round of amplification, where now we are taking the OR of events we want instead of the AND of events we don't want.

Let $\ell = n^\rho$, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = \frac{\log p_1}{\log p_2}$, and choose hash functions $g_1 \dots g_\ell$ independently as above. To store a point p, put it in a bucket for $g_j(p)$ for each j; these buckets are themselves stored in a hash table (by hashing the value of $g_j(p)$ down further) so that they fit in $O(n)$ space. Suppose now that $d(p, q) \leq r_1$ for some p. Then

$$\begin{aligned}
\Pr[g_j(p) = g_j(q)] &\geq p_1^k \\
&= p_1^{\log_{1/p_2} n} \\
&= n^{-\frac{\log 1/p_1}{\log 1/p_2}} \\
&= n^{-\rho} \\
&= 1/\ell.
\end{aligned}$$

So by searching through ℓ independent buckets we find p with probability

at least $1 - (1 - 1/\ell)^\ell \approx 1 - 1/e$. We'd like to guarantee that we only have to look at $O(n^\rho)$ points (most of which we may reject) during this process; but we can do this by stopping if we see more than 2ℓ points. Since we only expect to see ℓ bad points in all ℓ buckets, this event only happens with probability $1/2$. So even adding it to the probability of failure from the hash functions not working right we still have only a constant probability of failure.

Iterating the entire process $O(\log(1/\delta))$ times then gives the desired bound δ on the probability that this process fails to find a good point if one exists.

Multiplying out all the costs gives a cost of a query of $O(k\ell \log(1/\delta)) = O\left(n^\rho \log_{1/p_2} n \log(1/\delta)\right)$ hash function evaluations and $O(n^\rho \log(1/\delta))$ distance computations. The cost to insert a point is just $O(k\ell \log(1/\delta)) = O\left(n^\rho \log_{1/p_2} n \log(1/\delta)\right)$ hash function evaluations, the same number as for a query.

9.6.2.2 Hash functions for Hamming distance

Suppose that our points are d-bit vectors and that we use Hamming distance for our metric. In this case, using the family of one-bit projections $\{h_i : h_i(x) = x_i\}$ gives a locality-sensitive hash family [ABMRT96].

Specifically, we can show this family is $(r, r(1+\epsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\epsilon)}{d})$-sensitive. The argument is trivial: if two points p and q are at distance r or less, they differ in at most r places, so the probability that they hash together is just the probability that we don't pick one of these places, which is at least $1 - \frac{r}{d}$. Essentially the same argument works when p and q are far away.

These are not particularly clever hash functions, so the heavy lifting will be done by the (r_1, r_2)-PLEB construction. Our goal is to build an ϵ-PLEB for any fixed r, which will correspond to an $(r, r(1+\epsilon))$-PLEB. The main thing we need to do, following [IM98] as always, is compute a reasonable bound on $\rho = \frac{\log p_1}{\log p_2} = \frac{\ln(1-r/d)}{\ln(1-(1+\epsilon)r/d)}$. This is essentially just a matter of hitting it with enough inequalities, although there are a couple of tricks in the middle.

Compute

$$\begin{aligned}
\rho &= \frac{\ln(1 - r/d)}{\ln(1 - (1+\epsilon)r/d)} \\
&= \frac{(d/r)\ln(1 - r/d)}{(d/r)\ln(1 - (1+\epsilon)r/d)} \\
&= \frac{\ln((1 - r/d)^{d/r})}{\ln((1 - (1+\epsilon)r/d)^{d/r})} \\
&\leq \frac{\ln(e^{-1}(1 - r/d))}{\ln e^{-(1+\epsilon)}} \\
&= \frac{-1 + \ln(1 - r/d)}{-(1+\epsilon)} \\
&= \frac{1}{1+\epsilon} - \frac{\ln(1 - r/d)}{1+\epsilon}.
\end{aligned}$$

Note that we used the fact that $1 + x \leq e^x$ for all x in the denominator and $(1 - x)^{1/x} \geq e^{-1}(1 - x)$ for $x \in [0, 1]$ in the numerator.

We now pull a rabbit out of a hat by assuming that $r/d < 1/\ln n$.[4] This assumption can be justified by modifying the algorithm so that d is padded out with up to $d \ln n$ unused junk bits if necessary. Using the assumption, we get

$$\begin{aligned}
n^\rho &< n^{1/(1+\epsilon)} n^{-\ln(1-1/\ln n)/(1+\epsilon)} \\
&= n^{1/(1+\epsilon)}(1 - 1/\ln n)^{-\ln n} \\
&\leq e n^{1/(1+\epsilon)}.
\end{aligned}$$

Plugging into the formula for (r_1, r_2)-PLEB gives $O(n^{1/(1+\epsilon)} \log n \log(1/\delta))$ hash function evaluations per query, each of which costs $O(1)$ time, plus $O(n^{1/(1+\epsilon)} \log(1/\delta))$ distance computations, which will take $O(d)$ time each. If we add in the cost of the binary search, we have to multiply this by $O(\log \log R \log \log \log R)$, where the log-log-log comes from having to adjust δ so that the error doesn't accumulate too much over all $O(\log \log R)$ steps.

9.6.2.3 Hash functions for ℓ_1 distance

Essentially the same approach works for (bounded) ℓ_1 distance, using **discretization**, where we replace a continuous variable over some range with

[4] Indyk and Motwani pull this rabbit out of a hat a few steps earlier, but it's pretty much the same rabbit either way.

a discrete variable. Suppose we are working in $[0,1]^d$ with the ℓ_1 metric. Represent each coordinate x_i as a sequence of d/ϵ values x_{ij} in unary, for $j = 1 \ldots \epsilon d$, with $x_{ij} = 1$ if $\epsilon j/d < x_i$. Then the Hamming distance between the bit-vectors representing x and y is proportional to the ℓ_1 distance between the original vectors, plus an error term that is bounded by ϵ. We can then use the hash functions for Hamming distance to get a locality-sensitive hash family.

A nice bit about this construction is that we don't actually have to build the bit-vectors; instead, we can specify a coordinate x_i and a threshold c and get the same effect by recording whether $x_i > c$ or not.

Chapter 10

Randomized distributed algorithms

A **distributed algorithm** is one that runs on multiple machines that communicate with each other in some way, typically via **message-passing** (an abstraction of packet-based computer networks) or **shared memory** (an abstraction of multi-core CPUs and systems with a common memory bus). What generally distinguishes **distributed computing** from the closely-related idea of **parallel computing** is that we expect a lot of nondeterminism in a distributed algorithm: events take place at unpredictable times, processes may crash, and in particular bad cases we may have **Byzantine processes** that work deliberately against the algorithm.

This hostile nondeterminism is modeled by an **adversary** that controls scheduling and failures. For a shared-memory model, we have a collection of processes that each have a **pending operation** that is either a read or write to some **register**. The adversary chooses which of these pending operations happens next (so that concurrency between processes is modeled by interleaving their operations). Unlike the adversary that supplies the worst-case input to a traditional algorithm before it executes, an adversary scheduler might be able to observe the execution of a distributed algorithm in progress and adjust its choices in response. If it has full knowledge of the system (including internal states of processes), we call it an **adaptive adversary**; at the other extreme is an **oblivious adversary** that chooses the schedule of which processes execute operations at which times in advance.

As with traditional algorithms, distributed algorithms can use randomization to make themselves less predictable. The assumption is that even if the adversary is adaptive, it can't predict the outcome of future coin-flips.

For some problems, avoiding such predictability is provably necessary.

Distributed computing is a huge field, and we aren't going to be able to cover much of it in the limited space we have here. So we are going to concentrate on a simple problem that gives some of the flavor or randomized distributed algorithms (and that leads to problems that are still open).

10.1 Randomized consensus

The **consensus** problem is to get a collection of n processes to agree on a value. The requirements are that all the processes that don't crash finish the protocol (with probability 1 for randomized protocols) (**termination**), that they all output the same value (**agreement**), and that this value was an input to one of the processes (**validity**)—this last condition excludes protocols that always output the same value no matter what happens during the execution, and makes consensus useful for choosing among values generated by some other process.

10.1.1 Impossibility of deterministic algorithms

It's long been known that there are no deterministic algorithms for consensus in either a message-passing [FLP85] or shared-memory [LAA87] system. There is a simple intuitive proof of this result when all but one of the processes may crash (the previously-mentioned results assume at most one process may crash). Crash all but two processes, one with input 0 and one with input 1. Define the **preference** of a process as the value it will decided in a solo execution (this is well-defined because the processes are deterministic). In the initial state, the process with input b has preference b because of validity—it doesn't know the other process exists. But before it returns b, it has to cause the other process to change its preference (once it leaves the other process is running alone). It can do so only by sending a message or writing a register. When it is about to do this, stop it and run the other process until it does the same thing.

Either the other process somehow neutralizes the effect of the delayed message/write during this time, or it doesn't. In the first case, restart the first process (which still has its original preference and now must do something else to make the other process change). In the second case, deliver both operations and let the processes exchange preferences. The resulting execution looks very much like when two people are trying to pass each other in a hallway and oscillate back-and-forth—but since our process's have their

timing controlled by an adversary, the natural damping or randomness that occurs in humans doesn't ever resolve the situation.

10.1.2 Bounds on randomized algorithms

Adding randomness makes consensus possible, but it can still be expensive (some early protocols relied on all n processes flipping a coin the same way at the same time [Abr88]; naturally, this could take a wile for large n). With a shared-memory system and an adaptive adversary, consensus is expensive even with an optimal randomized algorithm: it costs $\Theta(n^2)$ total operations on average to reach agreement [AC08]. With an oblivious adversary (and shared memory), the situation is less clear. There exist two-valued consensus protocols in which each process does at most $O(\log n)$ operations on average and the processes together do at most $O(n)$ operations on average, but there are no good lower bounds showing that the $O(\log n)$ is necessary. We will describe a closely-related problem that shows where these bounds come from, and gives some hints to how they might (or might not) be improved.

10.2 Conciliators

A weakened version of consensus replaces the agreement requirement with **probabilistic agreement**: it's OK if the processes disagree sometimes as long as they agree with constant probability despite interference by the adversary. An algorithm that satisfies termination, validity, and probabilistic agreement is called a **conciliator**, and there are known techniques for turning a conciliator into a full consensus algorithm [Asp10].

```
   shared data: register r, initially ⊥
 1 k ← 0
 2 while r = ⊥ do
 3     with probability 2^k/(2n) do
 4         write v to r
 5     else
 6         do nothing
 7     end
 8     k ← k + 1
 9 end while
10 return r
```

Algorithm 10.1: Impatient first-mover conciliator from [Asp10]

Algorithm 10.1 implements a conciliator using a single register; it works against an oblivious adversary. The basic idea is that processes alternate between reading a register r and (maybe) writing to the register; if a process reads a non-null value from the register, it returns it. Any other process that reads the same non-null value will agree with the first process; the only way that this can't happen is if some process writes a different value to the register before it notices the first write.

The random choice of whether to write the register or not avoids this problem. The idea is that even though the adversary can schedule a write at a particular time, because it's oblivious, it won't be able to tell if the process wrote (or was about to write) or did a no-op instead.

The basic version of this algorithm, due to Chor, Israel, and Li [CIL94], uses a fixed $\frac{1}{2n}$ probability of writing to the register. So once some process writes to the register, the chance that any of the remaining $n-1$ processes write to it before noticing that it's non-null is at most $\frac{n-1}{2n} < 1/2$. It's also not hard to see that this algorithm uses $O(n)$ total operations, although it may be that one single process running by itself has to go through the loop $2n$ times before it finally writes the register and escapes.

Using increasing probabilities avoids this problem, because any process that executes the main loop $\lceil \lg n \rceil + 1$ times will write the register. This establishes the $O(\log n)$ per-process bound on operations. At the same time, an $O(n)$ bound on total operations still holds, since each write has at least a $\frac{1}{2n}$ chance of succeeding. The price we pay for the improvement is that we increase the chance that an initial value written to the register gets overwritten by some high-probability write. But the intuition is that the probabilities can't grow too much, because the probability that I write on my next write is close to the sum of the probabilities that I wrote on my previous writes—suggesting that if I have a high probability of writing next time, I should have done a write already.

Formalizing this intuition requires a little bit of work. Fix the schedule, and let p_i be the probability that the i-th write operation in this schedule succeeds. Let t be the least value for which $\sum_{i=1}^{t} p_i \geq 1/4$. We're going to argue that with constant probability one of the first t writes succeeds, and that the next $n-1$ writes by different processes all fail.

The probability that none of the first t writes succeed is

$$\begin{aligned}\prod_{i=1}^{t}(1-p_i) &\leq \prod_{i=1}^{t} e^{-p_i} \\ &= \exp\left(\sum_{i=1}^{t} p_i\right) \\ &\leq e^{-1/4}.\end{aligned}$$

Now observe that if some process q writes at or before the t-th write, then any process with a pending write either did no writes previously, or its last write was among the first $t-1$ writes, whose probabilities sum to less than $1/4$. In the first case, the process has a $\frac{1}{2n}$ chance of writing on its next attempt. In the second, it has a $\sum_{i\in S_q} p_i + \frac{1}{2n}$ chance of writing on its next attempt, where S_q is the set of indices in $1 \ldots t-1$ where q attempts to write.

Summing up these probabilities over all processes gives a total of $\frac{n-1}{2n} + \sum_q \sum_{i\in S_q} p_i \leq 1/2 + 1/4 = 3/4$. So with probabililty at least $e^{-1/4}(1 - 3/4) = e^{-1/4}/4$, we get agreement.

10.2.1 One-register lower bound

If we restrict ourselves to algorithms that use one register, there is a almost-matching lower bound: given any one-register conciliator, there is an oblivious adversary strategy that causes one process to do $\Omega(\log n/\log\log n)$ steps and all processes together to do $\Omega(n)$ steps, with at least constant probability. I found this result too late to include it in [Asp10], but assuming a single register is pretty restrictive, so it's not clear how publishable a result it is anyway. The lower bound assumes all processes have one of two inputs, 0 and 1; this is a more restrictive model that the one for the upper bound, which works for arbitrarily many values, and this may partially explain the small gap between the upper and lower bounds.

The adversary strategy for the lower bound proceeds in rounds. In each round, the adversary looks at the probability p_i that each process i writes to the register on its next operation, assuming things have gone well so far. For the first round, this is just the probability that the algorithm for process i chooses to write on its first step; since we assume that the adversary knows the algorithm, it can compute this probability easily. For subsequent rounds, it's a bit more complicated, since we may have to condition on what the process does and sees in previous rounds. But we will arrange things

so that these probabilities are needed only if nobody has yet written to the register, which means that the adversary can compute the probability that a process's j-th step is a write by simulating the process conditioned on its not doing a write earlier and not reading any value but $\perp$.

Having computed these probabilities, if there is a high probability that at least one process with input 0 (say) writes the register on its next operation, the adversary lets the input-1 processes run to completion (causing them to decide 1 because of validity) and then lets the input-0 processes run, with a good chance of overwriting all the work of the input-1 processes and violating agreement. Alternatively, if the input-0 processes are unlikely to write the register, we can just let them go ahead an accomplish nothing (possibly after crashing a few of them to get the probability low enough). A similar approach works for the input-1 processes.

Specifically, the adversary budgets an ϵ probability per round that it gets a bad outcome from the processes with a given input where either a process it needed to write doesn't or a process it needed not to write does. This bad outcome can happen in one of two ways:

1. The adversary allows all the process with some input b to run to completion, and then the other processes with input $\neg b$ don't overwrite the register. To avoid this we need $\Pr[\text{no overwrite}] = \prod_{i \in S_{\neg b}}(1 - p_i) \leq \epsilon$.

2. The adversary lets some processes in $S' \subseteq S_{\neg b}$ go first, but despite cutting down the numbers one of these processes writes. This occurs with probability $1 - \prod i \in S'(1 - p_i)$, and so if the adversary lets these processes go first, it will need to choose S' so that this quantity will be at most ϵ.

The adversary's goal in the second case is to maximize the size of S'. The algorithm's goal is to minimize the size of S', while at the same time not allowing the adversary to switch to the first case. It's not hard to see that the adversary does best by making S' consist of the processes with a given input that are least likely to write.

Suppose the k processes with a given input are ordered so that $p_1 \leq p_2 \leq \ldots p_k$. The algorithm is trying to maximize the least value t such that $\prod_{i=1}^{t}(1 - p_i) \geq 1 - \epsilon$ while at the same time keeping $\prod_{i=1}^{k}(1 - p_i) \leq \epsilon$. For any fixed t, the first quantity is maximized subject to the constraint by making all p_i equal (to prove this formally, use Lagrange multipliers). So the algorithm's best approach is to pick some fixed probability p for all writes such that $(1 - p)^k = \epsilon$, which gives $p = 1 - \epsilon^{1/k}$.

We can now compute how big S' can get as a function of ϵ and k. Let k' be the size of S'. Then $(1-p)^{k'} \geq 1-\epsilon$ gives $\epsilon^{k'/k} \geq 1-\epsilon$. Taking logs, we have $(k'/k)\ln\epsilon \geq \ln(1-\epsilon)$ or $k'/k \leq \frac{\ln(1-\epsilon)}{\ln\epsilon}$. This means that we should plan on keeping a fixed fraction of the processes with each input at each round, where the ratio depends on ϵ.

So now let's pick a value for ϵ. The constraints are that the number of rounds r will be $\log_{\frac{\ln\epsilon}{\ln(1-\epsilon)}}(n/2)$, and that we want $2r\epsilon$—the total probability that the adversary loses during the first r rounds—to be reasonably small (say less than $1/2$), so that we get an expected $r/2$ rounds even if we might finish early. Since we can't get a bound r better than $\Omega(\log n)$, a natural guess for ϵ would be $\Theta(1/\log n)$. Then $\ln\epsilon = \Theta(-\log\log n)$ and $\ln(1-\epsilon) = \Theta(-1/\log n)$, giving

$$\begin{aligned} r &= \frac{\ln n}{\ln\left(\frac{\ln\epsilon}{\ln(1-\epsilon)}\right)} \\ &= \frac{\ln n}{\ln\left(\frac{\Theta(-\log\log n)}{\Theta(-1/\log n)}\right)} \\ &= \Theta\left(\frac{\log n}{\log(\log n \log\log n)}\right) \\ &= \Theta\left(\frac{\log n}{\log\log n}\right). \end{aligned}$$

I don't know if this is the right answer or not. There is enough slop in the proof that I could easily believe that it should be $\Omega(\log n)$, but I could also believe that the $\log\log n$ in the denominator is a consequence of assuming only two possible input values.

10.2.2 An open problem

We have a one-register conciliator that is asymptotically optimal up to a $\log\log$ factor. What happens if we have two (or many) registers? The overwriting trick stops working, since there is no reason to believe that the 0-input processes would choose to write to the same registers as the 1-input processes. Instead, we'd probably need to make an argument based on the fact that writes to different registers commute in the sense that we can't tell which came first, meaning that you and I can't both have a pending operation that wins the game. But even if we can make this argument, we also have to deal with the fact that the oblivious adversary can't see what is going on. This was not so bad with one register, because in the lower

bound we carefully structured things so that the adversary could predict at least the distribution of the processes' behavior based on knowing that they couldn't see anything but $\perp$ in the register. But with multiple registers there doesn't seem to be any way to guarantee this, since there is no penalty for having a high probability of writing to a register that doesn't collide with anybody else.

Getting either an improved lower bound or improved upper bound here would tighten the bounds on the cost of randomized shared-memory consensus with an oblivious adversary. At the moment the lower bounds we have on oblivious-adversary consensus are still pretty weak; the best currently known bounds, due to Attiya and Censor-Hillel [ACH10], show that the running time for consensus has no better than a geometric distribution, but the expectation of this distribution works out to only $\Omega(1)$ operations per process.

Appendix A

Assignments

Assignments are typically due every other Wednesday at 17:00. Assignments may be turned in by placing them in Daniel Holtmann-Rice's mailbox in Arthur K. Watson Hall.

A.1 Assignment 1: due Wednesday, 2011-01-26, at 17:00

A.1.1 Bureaucratic part

Send me email! My address is `aspnes@cs.yale.edu`.

In your message, include:

1. Your name.
2. Your status: whether you are an undergraduate, grad student, auditor, etc.
3. Anything else you'd like to say.

(You will not be graded on the bureaucratic part, but you should do it anyway.)

A.1.2 Rolling a die

The usual model of a randomized algorithm assumes a source of fair, independent random bits. This makes it easy to generate uniform numbers in the range $0 \ldots 2^n - 1$, but not so easy for other ranges. Here are two algorithms for generating a uniform random integer $0 \leq s < n$:

- **Rejection sampling** generates a uniform random integer $0 \le s < 2^{\lceil \lg n \rceil}$. If $s < n$, return s; otherwise keep trying until you get some $s < n$.

- **Arithmetic coding** or **range coding** generates a sequence of bits $r_1, r_2, \ldots r_k$ until the half-open interval $[\sum_{i=1}^{k} 2^{-i} r_i, \sum_{i=1}^{k} 2^{-i} r_i + 2^{-k-1})$ is a subset of $[s/n, (s+1)/n)$ for some s; it then returns s.

1. Show that both rejection sampling and range coding produce a uniform value $0 \le s < n$ using an expected $O(\log n)$ random bits.

2. Which algorithm has a better constant?

3. Does there exist a function f and an algorithm that produces a uniform value $0 \le s < n$ for any n using $f(n)$ random bits with probability 1?

Solution

1. For rejection sampling, each sample requires $\lceil \lg n \rceil$ bits and is accepted with probability $n/2^{\lceil \lg n \rceil} \ge 1/2$. So rejection sampling returns a value after at most 2 samples on average, using no more than an expected $2 \lceil \lg n \rceil < 2(\lg n + 1)$ expected bits for the worst n.

   For range coding, we keep going as long as one of the $n-1$ nonzero endpoints s/n lies inside the current interval. After k bits, the probability that one of the 2^k intervals contains an endpoint is at most $(n-1)2^{-k}$; in particular, it drops below 1 as soon as $k = 2^{\lceil \lg n \rceil}$ and continues to drop by $1/2$ for each additional bit, requiring 2 more bits on average. So the expected cost of range coding is at most $\lceil \lg n \rceil + 2 < \lg n + 3$ bits.

2. We've just shown that range coding beats rejection sampling by a factor of 2 in the limit, for worst-case n. It's worth noting that other factors might be more important if random bits are cheap: rejection sampling is much easier to code and avoids the need for division.

3. There is no algorithm that produces a uniform value $0 \le s < n$ for all n using any fixed number of bits. Suppose such an algorithm existed. Fix some n. For all n values s to be equally likely, the sets of random bits $M^{-1}(s) = \{r | M(r) = s\}$ must have the same size. But this can only happen if n divides $2^{f(n)}$, which works only for n a power of 2.

A.1.3 Rolling many dice

Suppose you repeatedly roll an n-sided die. Give an asymptotic (big-Θ) bound on the expected number of rolls until you roll some number you have already rolled before.

Solution

In principle, it is possible to compute this value exactly, but we are lazy.

For a lower bound, observe that after m rolls, each of the $\binom{m}{2}$ pairs of rolls has probability $1/n$ of being equal, for an expected total of $\binom{m}{2}/n$ duplicates. For $m = \sqrt{n}/2$, this is less than $1/8$, which shows that the expected number of rolls is $\Omega(\sqrt{n})$.

For the upper bound, suppose we have already rolled the die $\sqrt{n}$ times. If we haven't gotten a duplicate already, each new roll has probability at least $\sqrt{n}/n = 1/\sqrt{n}$ of matching a previous roll. So after an additional $\sqrt{n}$ rolls on average, we get a repeat. This shows that the expected number of rolls is $O(\sqrt{n})$.

Combining these bounds shows that we need $\Theta(\sqrt{n})$ rolls on average.

A.1.4 All must have candy

A set of n_0 children each reach for one of n_0 candies, with each child choosing a candy independently and uniformly at random. If a candy is chosen by exactly one child, the candy and child drop out. The remaining n_1 children and candies then repeat the process for another round, leaving n_2 remaining children and candies, etc. The process continues until ever child has a candy.

Give the best bound you can on the expected number of rounds until every child has a candy.

Solution

Let $T(n)$ be the expected number of rounds remaining given we are starting with n candies. We can set up a probabilistic recurrence relation $T(n) = 1 + T(n - X_n)$ where X_n is the number of candies chosen by eactly one child. It is easy to compute $\mathrm{E}[X_n]$, since the probability that any candy gets chosen exactly once is $n(1/n)(1 - 1/n)^{n-1} = (1 - 1/n)^{n-1}$. Summing over all candies gives $\mathrm{E}[X_n] = n(1 - 1/n)^{n-1}$.

The term $(1-1/n)^{n-1}$ approaches e^{-1} in the limit, so for any fixed $\epsilon > 0$, we have $n(1 - 1/n)^{n-1} \geq n(e^{-1} - \epsilon)$ for sufficiently large n. We can get a quick bound by choosing ϵ so that $e^{-1} - \epsilon \geq 1/4$ (for example) and then

applying the Karp-Upfal-Wigderson inequality (2.3.1) with $\mu(n) = n/4$ to get

$$\begin{aligned} \mathrm{E}[T(n) &\leq \int_1^n \frac{1}{t/4}\,dt \\ &= 4 \ln n. \end{aligned}$$

There is a sneaky trick here, which is that we stop if we get down to 1 candy instead of 0. This avoids the usual problem with KUW and $\ln 0$, by observing that we can't ever get down to exactly one candy: if there were exactly one candy that gets grabbed twice or not at all, then there must be some other candy that also gets grabbed twice or not at all.

This analysis is sufficient for an asymptotic estimate: the last candy gets grabbed in $O(\log n)$ rounds on average. For most computer-sciency purposes, we'd be done here.

We can improve the constant slightly by observing that $(1 - 1/n)^{n-1}$ is in fact always greater than or equal to e^{-1}. The easiest way to see this is to plot the function, but if we want to prove it formally we can show that $(1 - 1/n)^{n-1}$ is a decreasing function by taking the derivative of its logarithm:

$$\begin{aligned} \frac{d}{dn} \ln(1 - 1/n)^{n-1} &= \frac{d}{dn}(n-1)\ln(1-1/n) \\ &= \ln(1-1/n) + \frac{n-1}{1-1/n} \cdot \frac{-1}{n^2}. \end{aligned}$$

and observing that it is negative for $n > 1$ (we could also take the derivative of the original function, but it's less obvious that it's negative). So if it approaches e^{-1} in the limit, it must do so from above, implying $(1-1/n)^{n-1} \geq e^{-1}$.

This lets us apply (2.3.1) with $\mu(n) = n/e$, giving $\mathrm{E}[T(n)] \leq e \ln n$.

If we skip the KUW bound and use the analysis in Section 2.4.2 instead, we get that $\Pr[T(n) \geq \ln n + \ln(1/\epsilon)] \leq \epsilon$. This suggests that the actual expected value should be $(1 + o(1)) \ln n$.

A.2 Assignment 2: due Wednesday, 2011-02-09, at 17:00

A.2.1 Randomized dominating set

A **dominating set** in a graph $G = (V, E)$ is a set of vertices D such that each of the n vertices in V is either in D or adjacent to a vertex in D.

Suppose we have a d-regular graph, in which every vertex has exactly d neighbors. Let D_1 be a random subset of V in which each vertex appears with independent probability p. Let D be the union of D_1 and the set of all vertices that are not adjacent to any vertex in D_1. (This construction is related to a classic maximal independent set algorithm of Luby [Lub85], and has the desirable property in a distributed system of finding a dominating set in only one round of communication.)

1. What would be a good value of p if our goal is to minimize $\mathrm{E}\,|D|$, and what bound on $\mathrm{E}\,|D|$ does this value give?

2. For your choice of p above, what bound can you get on $\Pr[|D| - \mathrm{E}\,|D| \geq t]$?

Solution

1. First let's compute $\mathrm{E}\,|D|$. Let X_v be the indicator for the event that $v \in D$. Then $X_v = 1$ if either (a) v is in D_1, which occurs with probability p; or (b) v and all d of its neighbors are not in D_1, which occurs with probability $(1-p)^{d+1}$. Adding these two cases gives $\mathrm{E}[X_v] = p + (1-p)^{d+1}$ and thus

$$\mathrm{E}\,|D| = \sum_v \mathrm{E}[X_v] = n\left(p + (1-p)^{d+1}\right). \tag{A.2.1}$$

   We optimize $\mathrm{E}\,|D|$ in the usual way, by seeking a minimum for $\mathrm{E}[X_v]$. Differentiating with respect to p and setting to 0 gives $1 - (d+1)(1-p)^d = 0$, which we can solve to get $p = 1 - (d+1)^{-1/d}$. (We can easily observe that this must be a minimum because setting p to either 0 or 1 gives $\mathrm{E}[X_v] = 1$.)

   The value of $\mathrm{E}\,|D|$ for this value of p is the rather nasty expression $n(1 - (d+1)^{-1/d} + (d+1)^{-1-1/d})$.

   Plotting the d factor up suggests that it goes to $\ln d/d$ in the limit, and both Maxima and `www.wolframalpha.com` agree with this. Knowing

the answer, we can prove it by showing

$$\begin{aligned}
\lim_{d\to\infty} \frac{1-(d+1)^{-1/d}+(d+1)^{-1-1/d}}{\ln d/d} &= \lim_{d\to\infty} \frac{1-d^{-1/d}+d^{-1-1/d}}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{1-d^{-1/d}}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{1-e^{-\ln d/d}}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{1-\left(1-\ln d/d+O(\ln^2 d/d^2)\right)}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{\ln d/d+O(\ln^2 d/d^2)}{\ln d/d} \\
&= \lim_{d\to\infty} (1+O(\ln d/d)) \\
&= 1.
\end{aligned}$$

This lets us write $\mathrm{E}\,|D| = (1+o(1))n \ln d/d$, where we bear in mind that the $o(1)$ term depends on d but not n.

2. Suppose we fix X_v for all but one vertex u. Changing X_u from 0 to 1 can increase $|D|$ by at most one (if u wasn't already in D) and can decrease it by at most $d-1$ (if u wasn't already in D and adding u to D_1 lets all d of u's neighbors drop out). So we can apply the method of bounded differences with $c_i = d-1$ to get

$$\Pr[|D| - \mathrm{E}\,|D| \geq t] \leq \exp\left(-\frac{t^2}{2n(d-1)^2}\right).$$

A curious feature of this bound is that it doesn't depend on p at all. It may be possible to get a tighter bound using a better analysis, which might pay off for very large d (say, $d \gg \sqrt{n}$).

A.2.2 Chernoff bounds with variable probabilities

Let $X_1 \ldots X_n$ be a sequence of 0–1 random variables, where for all i, $\mathrm{E}[X_i | X_1 \ldots X_{i-1}] \leq p_i$. Let $S = \sum_{i=1}^n X_i$ and $\mu = \sum_{i=1}^n p_i$. Show that, for all $\delta \geq 0$,

$$\Pr\left[S \geq (1+\delta)\mu\right] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu.$$

Solution

Let $S_t = \sum_{i=1}^{t} X_i$, so that $S = S_n$, and let $\mu_t = \sum_{i=1}^{t} p_i$. We'll show by induction on t that $\mathrm{E}[e^{\alpha S_t}] \leq \exp(e^{\alpha-1}\mu_t)$, when $\alpha > 0$.

Compute

$$\begin{aligned}
\mathrm{E}[e^{\alpha S}] &= \mathrm{E}[e^{\alpha S_{n-1}} e^{\alpha X_n}] \\
&= \mathrm{E}[e^{\alpha S_{n-1}} \mathrm{E}[e^{\alpha X_n} | X_1, \ldots, X_{n-1}]] \\
&= \mathrm{E}\left[e^{\alpha S_{n-1}} \left(\Pr[X_n = 0 | X_1, \ldots, X_{n-1}] + e^{\alpha} \Pr[X_n = 1 | X_1, \ldots, X_{n-1}]\right)\right] \\
&= \mathrm{E}\left[e^{\alpha S_{n-1}} \left(1 + +(e^{\alpha} - 1) \Pr[X_n = 1 | X_1, \ldots, X_{n-1}]\right)\right] \\
&\leq \mathrm{E}\left[e^{\alpha S_{n-1}} \left(1 + +(e^{\alpha} - 1) p_n\right)\right] \\
&\leq \mathrm{E}\left[e^{\alpha S_{n-1}} \exp\left((e^{\alpha} - 1) p_n\right)\right] \\
&\leq \mathrm{E}\left[e^{\alpha S_{n-1}}\right] \exp\left((e^{\alpha} - 1) p_n\right) \\
&\leq \exp\left(e^{\alpha} - 1\right) \mu_{n-1}) \exp\left((e^{\alpha} - 1) p_n\right) \\
&= \exp\left(e^{\alpha} - 1\right) \mu_n) .
\end{aligned}$$

Now apply the rest of the proof of (3.2.2) to get the full result.

A.2.3 Long runs

Let W be a binary string of length n, generated uniformly at random. Define a **run** of ones as a maximal sequence of contiguous ones; for example, the string 11100110011111101011 contains 5 runs of ones, of length 3, 2, 6, 1, and 2.

Let X_k be the number of runs in W of length k or more.

1. Compute the exact value of $\mathrm{E}[X_k]$ as a function of n and k.
2. Give the best concentration bound you can for $|X_k - \mathrm{E}[X_k]|$.

Solution

1. We'll compute the probability that any particular position $i = 1 \ldots n$ is the start of a run of length k or more, then sum over all i. For a run of length k to start at position i, either (a) $i = 1$ and $W_i \ldots W_{i+k-1}$ are all 1, or (b) $i > 1$, $W_{i-1} = 0$, and $W_i \ldots W_{i+k-1}$ are all 1. Assuming $n \geq k$, case (a) adds 2^{-k} to $\mathrm{E}[X_k]$ and case (b) adds $(n-k)2^{-k-1}$, for a total of $2^{-k} + (n-k)2^{-k-1} = (n-k+2)2^{-k-1}$.

2. We can get an easy bound without too much cleverness using McDiarmid's inequality (3.2.17). Observe that X_k is a function of the independent random variables $W_1 \dots W_n$ and that changing one of these bits changes X_k by at most 1 (this can happen in several ways: a previous run of length $k-1$ can become a run of length k or vice versa, or two runs of length k or more separated by a single zero may become a single run, or vice versa). So (3.2.17) gives $\Pr[|X - \mathrm{E}[X]| \geq t] \leq 2\exp\left(-\frac{t^2}{2n}\right)$.

   We can improve on this a bit by grouping the W_i together into blocks of length ℓ. If we are given control over a block of ℓ consecutive bits and want to minimize the number of runs, we can either (a) make all the bits zero, causing no runs to appear within the block and preventing adjacent runs from extending to length k using bits from the block, or (b) make all the bits one, possibly creating a new run but possibly also causing two existing runs on either side of the block to merge into one. In the first case, changing all the bits to one except for a zero after every k consecutive ones creates at most $\left\lfloor \frac{\ell+2k-1}{k+1} \right\rfloor$ new runs. Treating each of the $\lceil n \rceil \ell$ blocks as a single variable then gives $\Pr[|X - \mathrm{E}[X]| \geq t] \leq 2\exp\left(-\frac{t^2}{2\lceil n/\ell \rceil (\lfloor (\ell+2k-1)/(k+1) \rfloor)^2}\right)$. Staring at plots of the denominator for a while suggests that it is minimized at $\ell = k+3$, the largest value with $\lfloor (\ell + 2k - 1)/(k+1) \rfloor \leq 2$. This gives $\Pr[|X - \mathrm{E}[X]| \geq t] \leq 2\exp\left(-\frac{t^2}{8\lceil n/(k+3) \rceil}\right)$, improving the bound on t from $\Theta(\sqrt{n \log(1/\epsilon)})$ to $\Theta(\sqrt{(n/k)\log(1/\epsilon)})$.

   For large k, the expectation of any individual X_k becomes small, so we'd expect that Chernoff bounds would work better on the upper bound side than the method of bounded differences. Unfortunately, we don't have independence. But from Problem A.2.2, we know that the usual Chernoff bound works as long as we can show $\mathrm{E}[X_i | X_1, \dots, X_{i-1}] \leq p_i$ for some sequence of fixed bounds p_i.

   For X_1, there are no previous X_i, and we have $\mathrm{E}[X_1] = 2^{-k}$ exactly.

   For X_i with $i > 1$, fix $X_1, \dots, X_{i-1}$; that is, condition on the event $X_j = x_j$ for all $j < i$ with some fixed sequence $x_1, \dots, x_{i-1}$. Let's call this event A. Depending on the particular values of the x_j, it's not clear how conditioning on A will affect X_i; but we can split on the

value of W_{i-1} to show that either it has no effect or $X_i = 0$:

$$\begin{aligned}\mathrm{E}[X_i|A] &= \mathrm{E}[X_i|A, W_{i-1}=0]\Pr[W_{i-1}=0|A] + \mathrm{E}[X_i|A, W_{i-1}=1]\Pr[W_{i-1}=1|A] \\ &\le 2^{-k}\Pr[W_{i-1}=0|A] + 0 \\ &\le 2^{-k}.\end{aligned}$$

So we have $p_i \le 2^{-k}$ for all $1 \le i \le n-k+1$. This gives $\mu = 2^{-k}(n-k+1)$, and $\Pr[X \ge (1+\delta)\mu] \le \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu$.

If we want a two-sided bound, we can set $\delta = 1$ (since X can't drop below 0 anyway, and get $\Pr[|X - \mathrm{E}[X]| > 2^{-k}(n-k+1)] \le \left(\frac{e}{4}\right)^{2^{-k}(n-k+1)}$. This is exponentially small for $k = o(\lg n)$. If k is much bigger than $\lg n$, then we have $\mathrm{E}[X] \ll 1$, so Markov's inequality alone gives us a strong concentration bound.

However, in both cases, the bounds are competitive with the previous bounds from McDiarmid's inequality only if $\mathrm{E}[X] = O(\sqrt{n\log(1/\epsilon)})$. So McDiarmid's inequality wins for $k = o(\log n)$, Markov's inequality wins for $k = \omega(\log n)$, and Chernoff bounds may be useful for a small interval in the middle.

A.3 Assignment 3: due Wednesday, 2011-02-23, at 17:00

A.3.1 Longest common subsequence

A **common subsequence** of two sequences v and w is a sequence u of length k such that there exist indices $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_k$ with $u_\ell = v_{i_\ell} = w_{j_\ell}$ for all ℓ. For example, `ardab` is a common subsequence of `abracadabra` and `cardtable`.

Let v and w be words of length n over an alphabet of size n drawn independently and uniformly at random. Give the best upper bound you can on the expected length of the longest common subsequence of v and w.

Solution

Let's count the expectation of the number X_k of common subsequences of length k. We have $\binom{n}{k}$ choices of positions in v, and $\binom{n}{k}$ choices of positions in w; for each such choices, there is a probability of exactly n^{-k} that the

corresponding positions match. This gives

$$\begin{aligned}\mathrm{E}[X_k] &= \binom{n}{k}^2 n^{-k} \\ &< \frac{n^k}{(k!)^2}. \\ &< \frac{n^k}{(k/e)^{2k}} \\ &= \left(\frac{ne^2}{k^2}\right)^k.\end{aligned}$$

We'd like this bound to be substantially less than 1. We can't reasonably expect this to happen unless the base of the exponent is less than 1, so we need $k > e\sqrt{n}$.

If $k = (1+\epsilon)e\sqrt{n}$ for any $\epsilon > 0$, then $\mathrm{E}[X_k] < (1+\epsilon)^{-2e\sqrt{n}} < \frac{1}{n}$ for sufficiently large n. It follows that the expected length of the longest common subsequent is at most $(1+\epsilon)e\sqrt{n}$ for sufficiently large n (because if there are no length-k subsequences, the longest subsequence has length at most $k-1$, and if there is at least one, the longest has length at most n; this gives a bound of at most $(1-1/n)(k-1) + (1/n)n < k$). So in general we have the length of the longest common subsequence is at most $(1+o(1))e\sqrt{n}$.

Though it is not required by the problem, here is a quick argument that the expected length of the longest common subsequence is $\Omega(\sqrt{n})$, based on the **Erdős-Szekeres theorem** [ES35].[1] The Erdős-Szekeres theorem says that any permutation of n elements contains either an increasing sequence of $\sqrt{n}$ elements or a decreasing sequence of $\sqrt{n}$ elements. Given two random sequences of length n, let S be the set of all elements that appear in both, and consider two permutations ρ and σ of S corresponding to the order in which the elements appear in v and w, respectively (if an element appears multiple times, pick one of the occurrences at random). Then the Erdős-Szekeres theorem says that ρ contains a sequence of length $\sqrt{|\rho|}$ that is either increasing or decreasing with respect to the order given by σ; by symmetry, the probability that it is increasing is at least $1/2$. This gives an expected value for the longest common subsequence that is at least $\mathrm{E}[\sqrt{|\rho|}]/2$.

Let $X = |\rho|$. We can compute a lower bound $\mathrm{E}[X]$ easily; each possible element fails to occur in v with probability $(1-1/n)^n \le e^{-1}$, and similarly for w. So the chance that an element appears in both sequences is at least

[1] As suggested by Benjamin Kunsberg.

$(1-e^{-1})^2$, and thus $\mathrm{E}[X] \geq n(1-e^{-1})^2$. What we want is $\mathrm{E}[\sqrt{X}]/2$; but here the fact that $\sqrt{x}$ is concave means that $\mathrm{E}[\sqrt{X}] \geq \sqrt{\mathrm{E}[X]}$ by Jensen's inequality (3.1.5). So we have $\mathrm{E}[\sqrt{|\rho|}]/2 \geq \sqrt{n(1-e^{-1})^2}/2 = \frac{1-e^{-1}}{2}\sqrt{n}$.

This is not a very good bound (empirically, the real bound seems to be in the neighborhood of $1.9\sqrt{n}$ when $n = 10000$), but it shows that the upper bound of $(1+o(1))e\sqrt{n}$ is tight up to constant factors.

A.3.2 A strange error-correcting code

Let Σ be an alphabet of size $m+1$ that includes m non-blank symbols and a special blank symbol. Let S be a set of $\binom{n}{k}$ strings of length n with non-blank symbols in exactly k positions each, such that no two strings in S have non-blank symbols in the same k positions.

For what value of m can you show S exists such that no two strings in S have the same non-blank symbols in $k-1$ positions?

Solution

This is a job for the Lovász local lemma. And it's even symmetric, so we can use the symmetric version (Corollary 4.3.2).

Suppose we assign the non-blank symbols to each string uniformly and independently at random. For each $A \subseteq S$ with $|A| = k$, let X_A be the string that has non-blank symbols in all positions in A. For each pair of subsets A, B with $|A| = |B| = k$ and $|A \cap B| = k-1$, let $C_{A,B}$ be the event that X_A and X_B are identical on all positions in $A \cap B$. Then $\Pr[C_{A,B}] = m^{-k+1}$.

We now now need to figure out how many events are in each neighborhood $\Gamma(C_{A,B})$. Since $C_{A,B}$ depends only on the choices of values for A and B, it is independent of any events $C_{A',B'}$ where neither of A' or B' is equal to A or B. So we can make $\Gamma(C_{A,B})$ consist of all events $C_{A,B'}$ and $C_{A',B}$ where $B' \neq B$ and $A' \neq A$.

For each fixed A, there are exactly $(n-k)k$ events B that overlap it in $k-1$ places, because we can specify B by choosing the elements in $B \setminus A$ and $A \setminus B$. This gives $(n-k)k-1$ events $C_{A,B'}$ where $B' \neq B$. Applying the same argument for A' gives a total of $d = 2(n-k)k-2$ events in $\Gamma(C_{A,B})$. Corollary 4.3.2 applies if $ep(d+1) \leq 1$, which in this case means $em^{-(k-1)}(2(n-k)k-1) \leq 1$. Solving for m gives

$$m \geq (2e(n-k)k-1)^{1/(k-1)}. \tag{A.3.1}$$

For $k \ll n$, the $(n-k)^{1/(k-1)} \approx n^{1/(k-1)}$ term dominates the shape of the right-hand side asymptotically as k gets large, since everything else goes

to 1. This suggests we need $k = \Omega(\log n)$ to get m down to a constant.

Note that (A.3.1) doesn't work very well when $k = 1$.[2] For the $k = 1$ case, there is no overlap in non-blank positions between different strings, so $m = 1$ is enough.

A.3.3 A multiway cut

Given a graph $G = (V, E)$, a **3-way cut** is a set of edges whose endpoints lie in different parts of a partition of the vertices V into three disjoint parts $S \cup T \cup U = V$.

1. Show that any graph with m edges has a 3-way cut with at least $2m/3$ edges.

2. Give an efficient deterministic algorithm for finding such a cut.

Solution

1. Assign each vertex independently to S, T, or U with probability $1/3$ each. Then the probability that any edge uv is contained in the cut is exactly $2/3$. Summing over all edges gives an expected $2m/3$ edges.

2. We'll derandomize the random vertex assignment using the method of conditional probabilities. Given a partial assignment of the vertices, we can compute the conditional expectation of the size of the cut assuming all other vertices are assigned randomly: each edge with matching assigned endpoints contributes 0 to the total, each edge with non-matching assigned endpoints contributes 1, and each edge with zero or one assigned endpoints contributes $2/3$. We'll pick values for the vertices in some arbitrary order to maximize this conditional expectation (since our goal is to get a large cut). At each step, we need only consider the effect on edges incident to the vertex we are assigning whose other endpoints are already assigned, because the contribution of any other edge is not changed by the assignment. Then maximizing the conditional probability is done by choosing an assignment that matches the assignment of the fewest previously-assigned neighbors: in other words, the natural greedy algorithm works. The cost of this algorithm is $O(n+m)$, since we loop over all vertices and have to check each edge at most once for each of its endpoints.

[2] Thanks to Brad Hayes for pointing this out.

A.4 Assignment 4: due Wednesday, 2011-03-23, at 17:00

A.4.1 Sometimes successful betting strategies are possible

You enter a casino with $X_0 = a$ dollars, and leave if you reach 0 dollars or b or more dollars, where $a, b \in \mathbb{N}$. The casino is unusual in that it offers arbitrary fair games subject to the requirements that:

- Any payoff resulting from a bet must be a nonzero integer in the range $-X_t$ to X_t, inclusive, where X_t is your current wealth.
- The expected payoff must be exactly 0. (In other words, your assets X_t should form a martingale sequence.)

For example, if you have 2 dollars, you may make a bet that pays off -2 with probability $2/5$, $+1$ with probability $2/5$ and $+2$ with probability $1/5$; but you may not make a bet that pays off -3, $+3/2$, or $+4$ under any circumstances, or a bet that pays off -1 with probability $2/3$ and $+1$ with probability $1/3$.

1. What strategy should you use to maximize your chances of leaving with at least b dollars?
2. What strategy should you use to maximize your changes of leaving with nothing?
3. What strategy should you use to maximize the number of bets you make before leaving?

Solution

1. Let X_t be your wealth at time t, and let τ be the stopping time when you leave. Because $\{X_t\}$ is a martingale, $\mathrm{E}[X_0] = a = \mathrm{E}[X_\tau] = \Pr[X_\tau \geq b]\,\mathrm{E}[X_\tau | X_\tau \geq b]$. So $\Pr[X_\tau \geq b]$ is maximized by making $\mathrm{E}[X_\tau | X_\tau \geq b]$ as small as possible. It can't be any smaller than b, which can be obtained exactly by making only ± 1 bets. The gives a probability of leaving with b of exactly a/b.
2. Here our goal is to minimize $\Pr[X_\tau \geq b]$, so we want to make $\mathrm{E}[X_\tau | X_\tau \geq b]$ as large as possible. The largest value of X_τ we can possibly reach is $2(b-1)$; we can obtain this value by betting ± 1 until we reach $b-1$, then making any fair bet with positive payoff $b-1$ (for example,

$\pm(b-1)$ with equal probability works, as does a bet that pays off $b-1$ with probability $1/b$ and -1 with probability $(b-1)/b$). In this case we get a probability of leaving with 0 of $1 - \frac{a}{2(b-1)}$.

3. For each t, let $\Delta_t = X_t - X_{t-1}$ and $V_t = \mathrm{Var}[\Delta_t|\mathcal{F}_t]$. We have previously shown (see the footnote to Section 6.4.1) that $\mathrm{E}[X_\tau^2] = \mathrm{E}[X_0^2] + \mathrm{E}[\sum_{t=1}^{\tau} V_t]$ where τ is an appropriate stopping time. When we stop, we know that $X_\tau^2 \leq (2(b-1))^2$, which puts an upper bound on $\mathrm{E}[\sum_{i=1}^{\tau} V_i]$. We can spend this bound most parsimoniously by minimizing V_i as much as possible. If we make each $\Delta_t = \pm 1$, we get the smallest possible value for V_t (since any change contributes at least 1 to the variance). However, in this case we don't get all the way out to $2(b-1)$ at the high end; instead, we stop at b, giving an expected number of steps equal to $a(b-a)$.

   We can do a bit better than this by changing our strategy at $b-1$. Instead of betting ± 1, let's pick some x and place a bet that pays off $b-1$ with probability $\frac{1}{b}$ and -1 with probability $\frac{b-1}{b} = 1 - \frac{1}{b}$. (The idea here is to minimize the conditional variance while still allowing ourselves to reach $2(b-1)$.) Each ordinary random walk step has $V_t = 1$; a "big" bet starting at $b-1$ has $V_t = 1 - \frac{1}{b} + \frac{(b-1)^2}{b} = \frac{b-1+b^2-2b+1}{b} = b - \frac{1}{b}$.

   To analyze this process, observe that starting from a, we first spending $a(b-a-1)$ steps on average to reach either 0 (with probability $1 - \frac{a}{b-1}$ or $b-1$ (with probability $\frac{a}{b-1}$. In the first case, we are done. Otherwise, we take one more step, then with probability $\frac{1}{b}$ we lose and with probability $\frac{b-1}{b}$ we continue starting from $b-2$. We can write a recurrence for our expected number of steps $T(a)$ starting from a, as:

$$T(a) = a(b-a-1) + \frac{a}{b-1}\left(1 + \frac{b-1}{b}T(b-2)\right). \qquad \text{(A.4.1)}$$

   When $a = b-2$, we get

$$\begin{aligned} T(b-2) &= (b-2) + \frac{b-2}{b-1}\left(1 + \frac{b-1}{b}T(b-2)\right) \\ &= (b-2)\left(1 + \frac{1}{b-1}\right) + \frac{b-2}{b}T(b-2), \end{aligned}$$

which gives

$$\begin{aligned} T(b-2) &= \frac{(b-2)\frac{2b-1}{b-1}}{2/b} \\ &= \frac{b(b-2)(2b-1)}{2(b-1)}. \end{aligned}$$

Plugging this back into (A.4.1) gives

$$\begin{aligned} T(a) &= a(b-a-1) + \frac{a}{b-1}\left(1 + \frac{b-1}{b}\frac{b(b-2)(2b-1)}{2(b-1)}\right) \\ &= ab - a^2 + a + \frac{a}{b-1} + \frac{a(b-2)(2b-1)}{2(b-1)} \\ &= \frac{3}{2}ab + O(b). \end{aligned} \tag{A.4.2}$$

This is much better than the $a(b-a)$ value for the straight ± 1 strategy, especially when a is also large.

I don't know if this particular strategy is in fact optimal, but that's what I'd be tempted to bet.

A.4.2 Random walk with reset

Consider a random walk on $\mathbb{N}$ that goes up with probability $1/2$, down with probability $3/8$, and resets to 0 with probability $1/8$. When $X_t > 0$, this gives:

$$X_{t+1} = \begin{cases} X_t + 1 & \text{with probability } 1/2, \\ X_t - 1 & \text{with probability } 3/8, \text{ and} \\ 0 & \text{with probability } 1/8. \end{cases}$$

When $X_t = 0$, we let $X_{t+1} = 1$ with probability $1/2$ and 0 with probability $1/2$.

1. What is the stationary distribution of this process?
2. What is the mean recurrence time μ_n for some state n?
3. Use μ_n to get a tight asymptotic (i.e., big-Θ) bound on $\mu_{0,n}$, the expected time to reach n starting from 0.

Solution

1. For $n > 0$, we have $\pi_n = \frac{1}{2}\pi_{n-1} + \frac{3}{8}\pi_{n+1}$, with a base case $\pi_0 = \frac{1}{8} + \frac{3}{8}\pi_0 + \frac{3}{8}\pi_1$.

   The π_n expression is a linear homogeneous recurrence, so its solution consists of linear combinations of terms b^n, where b satisfies $1 = \frac{1}{2}b^{-1} + \frac{3}{8}b$. The solutions to this equation are $b = 2/3$ and $b = 2$; we can exclude the $b = 2$ case because it would make our probabilities blow up for large n. So we can reasonably guess $\pi_n = a(2/3)^n$ when $n > 0$.

   For $n = 0$, substitute $\pi_0 = \frac{1}{8} + \frac{3}{8}\pi_0 + \frac{3}{8}a(2/3)$ to get $\pi_0 = \frac{1}{5} + \frac{2}{5}a$. Now substitute

$$\begin{aligned}\pi_1 &= (2/3)a \\ &= \frac{1}{2}\pi_0 + \frac{3}{8}a(2/3)^2 \\ &= \frac{1}{2}\left(\frac{1}{5} + \frac{2}{5}a\right) + \frac{3}{8}a(2/3)^2 \\ &= \frac{1}{10} + \frac{11}{30}a,\end{aligned}$$

   which we can solve to get $a = 1/3$.

   So our candidate π is $\pi_0 = 1/3$, $\pi_n = (1/3)(2/3)^n$, and in fact we can drop the special case for π_0.

   As a check, $\sum_{i=0}^{n} \pi_n = (1/3)\sum_{i=0}^{n}(2/3)^n = \frac{1/3}{1-2/3} = 1$.

2. Since $\mu_n = 1/\pi_n$, we have $\mu_n = 3(3/2)^n$.

3. In general, let $\mu_{k,n}$ be the expected time to reach n starting at k. Then $\mu_n = \mu_{n,n} = 1 + \frac{1}{8}\mu_{0,n} + \frac{1}{2}\mu_{n+1,n} + \frac{3}{8}\mu_{n-1,n} \geq 1 + \mu_{0,n}/8$. It follows that $\mu_{0,n} \leq 8\mu_n + 1 = 24(3/2)^n + 1 = O((3/2)^n)$.

   For the lower bound, observe that $\mu_n \leq \mu_{n,0} + \mu_{0,n}$. Since there is a $1/8$ chance of reaching 0 from any state, we have $\mu_{n,0} \leq 8$. It follows that $\mu_n \leq 8 + \mu_{0,n}$ or $\mu_{0,n} \geq \mu_n - 8 = \Omega((3/2)^n)$.

A.4.3 Yet another shuffling algorithm

Suppose we attempt to shuffle a deck of n cards by picking a card uniformly at random, and swapping it with the top card. Give the best bound you can on the mixing time for this process to reach a total variation distance of ϵ from the uniform distribution.

Solution

It's tempting to use the same coupling as for move-to-top (see Section 7.3.4). This would be that at each step we choose the same card to swap to the top position, which increases by at least one the number of cards that are in the same position in both decks. The problem is that at the next step, these two cards are most likely separated again, by being swapped with other cards in two different positions.

Instead, we will do something slightly more clever. Let Z_t be the number of cards in the same position at time t. If the top cards of both decks are equal, we swap both to the same position chosen uniformly at random. This has no effect on Z_t. If the top cards of both decks are not equal, we pick a card uniformly at random and swap it to the top in both decks. This increases Z_t by at least 1, unless we happen to pick cards that are already in the same position; so Z_t increases by at least 1 with probability $1 - Z_t/n$.

Let's summarize a state by an ordered pair (k, b) where $k = Z_t$ and b is 0 if the top cards are equal and 1 if they are not equal. Then we have a Markov chain where $(k, 0)$ goes to $(k, 1)$ with probability $\frac{n-k}{n}$ (and otherwise stays put); and $(k, 1)$ goes to $(k+1, 0)$ (or higher) with probability $\frac{n-k}{n}$ and to $(k, 0)$ with probability $\frac{k}{n}$.

Starting from $(k, 0)$, we expect to wait $\frac{n}{n-k}$ steps on average to reach $(k, 1)$, at which point we move to $(k + 1, 0)$ or back to $(k, 0)$ in one more step; we iterate through this process $\frac{n}{n-k}$ times on average before we are successful. This gives an expected number of steps to get from $(k, 0)$ to $(k + 1, 0)$ (or possibly a higher value) of $\frac{n}{n-k}\left(\frac{n}{n-k} + 1\right)\Big)$. Summing over k up to $n - 2$ (since once $k > n - 2$, we will in fact have $k = n$, since k can't be $n - 1$), we get

$$\begin{aligned}
\mathrm{E}[\tau] &\le \sum_{k=0}^{n-2} \frac{n}{n-k}\left(\frac{n}{n-k} + 1\right) \\
&= \sum_{m=2}^{n} \left(\frac{n^2}{m^2} + \frac{n}{m}\right) \\
&\le n^2\left(\frac{\pi^2}{6} - 1\right) + n \ln n. \\
&= O(n^2).
\end{aligned}$$

So we expect the deck to mix in $O(n^2 \log(1/\epsilon))$ steps. (I don't know if this is the real bound; my guess is that it should be closer to $O(n \log n)$ as in all the other shuffling procedures.)

A.5 Assignment 5: due Thursday, 2011-04-07, at 23:59

A.5.1 A reversible chain

Consider a random walk on $\mathbb{Z}_m$, where $p_{i,i+1} = 2/3$ for all i and $p_{i,i-1} = 1/3$ for all i except $i = 0$. Is it possible to assign values to $p_{0,m-1}$ and $p_{0,0}$ to make this chain reversible, and if so, what stationary distribution do you get?

Solution

Suppose we can make this chain reversible, and let π be the resulting stationary distribution. From the detailed balance equations, we have $(2/3)\pi_i = (1/3)\pi_{i+1}$ or $\pi_{i+1} = 2\pi_i$ for $i = 0 \ldots m-2$. The solution to this recurrence is $\pi_i = 2^i\pi_0$, which gives $\pi_i = \frac{2^i}{2^m-1}$ when we set π_0 to get $\sum_i \pi_i = 1$.

Now solve $\pi_0 p_{0,m-1} = \pi_{m-1}p_{m-1,0}$ to get

$$\begin{aligned} p_{0,m-1} &= \frac{\pi_{m-1}p_{m-1,0}}{\pi_0} \\ &= 2^{m-1}(2/3) \\ &= 2^m/3. \end{aligned}$$

This is greater than 1 for $m > 1$, so except for the degenerate cases of $m = 1$ and $m = 2$, it's not possible to make the chain reversible.

A.5.2 Toggling bits

Consider the following Markov chain on an array of n bits $a[1], a[2], \ldots a[n]$. At each step, we choose a position i uniformly at random. We then change $A[i]$ to $\neg A[i]$ with probability $1/2$, provided $i = 1$ or $A[i-1] = 1$ (if neither condition holds hold, do nothing).[3]

1. What is the stationary distribution?

2. How quickly does it converge?

[3]Motivation: Imagine each bit represents whether a node in some distributed system is inactive (0) or active (1), and you can only change your state if you have an active left neighbor to notify. Also imagine that there is an always-active *base station* at -1 (alternatively, imagine that this assumption makes the problem easier than the other natural arrangement where we put all the nodes in a ring).

Solution

1. First let's show irreducibility. Starting from an arbitrary configuration, repeatedly switch the leftmost 0 to a 1 (this is always permitted by the transition rules); after at most n steps, we reach the all-1 configuration. Since we can repeat this process in reverse to get to any other configuration, we get that every configuration is reachable from every other configuration in at most $2n$ steps ($2n-1$ if we are careful about handling the all-0 configuration separately).

   We also have that for any two adjacent configurations x and y, $p_{xy} = p_{yx} = \frac{1}{2n}$. So we have a reversible, irreducible, aperiodic (because there exists at least one self-loop) chain with a uniform stationary distribution $\pi_x = 2^{-n}$.

2. Here is a bound using the obvious coupling, where we choose the same position in X and Y and attempt to set it to the same value. To show this coalesces, given X_t and Y_t define Z_t to be the position of the rightmost 1 in the common prefix of X_t and Y_t, or 0 if there is no 1 in the common prefix of X_t and Y_t. Then Z_t increases by at least 1 if we attempt to set position $Z_t + 1$ to 1, which occurs with probability $\frac{1}{2n}$, and decreases by at most 1 if we attempt to set Z_t to 0, again with probability $\frac{1}{2n}$. It follows that Z_t reaches n no later than a ± 1 random walk on $0 \dots n$ with reflecting barriers that takes a step every $1/n$ time units on average. The expected number of steps to reach n from the worst-case starting position of 0 is exactly n^2. (Proof: model the random walk with a reflecting barrier at 0 by folding a random walk with absorbing barriers at $\pm n$ in half, then use the bound from Section 6.4.1.) We must then multiply this by n to get an expected n^3 steps in the original process. So the two copies coalesce in at most n^3 expected steps. My suspicion is one could improve this bound with a better analysis by using the bias toward increasing Z_t to get the expected time to coalesce down to $O(n^2)$, but I don't know any clean way to do this.

   The path coupling version of this is that we look at two adjacent configurations X_t and Y_t, use the obvious coupling again, and see what happens to $\mathrm{E}[d(X_{t+1}, Y_{t+1})|X_t, Y_t]$, where the distance is the number of transitions needed to convert X_t to Y_t or vice versa. If we pick the position i where X_t and Y_t differ, then we coalesce; this occurs with probability $1/n$. If we change the 1 to the left of i to a 0, then $d(X_{t+1}, Y_{t+1})$ rises to 3 (because to get from X_{t+1} to Y_{t+1}, we have to

change position $i-1$ to 1, change position i, and then change position $i-1$ back to 0); this occurs with probability $1/2n$ if $i > 1$. But we can also get into trouble if we try to change position $i+1$; we can only make the change in one of X_t and Y_t, so we get $d(X_{t+1}, Y_{t+1}) = 2$ in this case, which occurs with probability $1/2n$ when $i < n$. Adding up all three cases gives a worst-case expected change of $-1/n + 2/2n + 1/2n = 1/2n > 0$. So unless we can do something more clever, path coupling won't help us here.

However, it is possible to get a bound using canonical paths, but the best bound I could get was not as good as the coupling bound. The basic idea is that we will change x into y one bit at a time (from left to right, say), so that we will go through a sequence of intermediate states of the form $y[1]y[2]\ldots y[i]x[i+1]x[i+2]\ldots x[n]$. But to change $x[i+1]$ to $y[i+1]$, we may also need to reach out with a tentacle of 1 bits from from the last 1 in the current prefix of y (and then retract it afterwards). Given a particular transition where we change a 0 to a 1, we can reconstruct the original x and y by specifying (a) which bit i at or after our current position we are trying to change; (b) which 1 bit before our current position is the last "real" 1 bit in y as opposed to something we are creating to reach out to position i; and (c) the values of $x[1]\ldots x[i-1]$ and $y[i+1]\ldots y[i]$. A similar argument applies to 1–0 transitions. So we are routing at most $n^2 2^{n-1}$ paths across each transition, giving a bound on the congestion

$$\begin{aligned}\rho &\leq \left(\frac{1}{2^{-n}/2n}\right) n^2 2^{n-1} 2^{-2n} \\ &= n^3.\end{aligned}$$

The bound on τ_2 that follows from this is $8n^6$, which is pretty bad (although the constant could be improved by counting the (a) and (b) bits more carefully). As with the coupling argument, it may be that there is a less congested set of canonical paths that gives a better bound.

A.5.3 Spanning trees

Suppose you have a connected graph $G = (V, E)$ with n nodes and m edges. Consider the following Markov process. Each state H_t is a subgraph of G that is either a spanning tree or a spanning tree plus an additional edge. At each step, flip a fair coin. If it comes up heads, choose an edge e uniformly

at random from E and let $H_{t+1} = H_t \cup \{e\}$ if H_t is a spanning tree and let $H_{t+1} = H_t \setminus \{e\}$ if H_t is not a spanning tree and $H_t \setminus \{e\}$ is connected. If it comes up tails and H_t is a spanning tree, let H_{t+1} be some other spanning tree, sampled uniformly at random. In all other cases, let $H_{t+1} = H_t$.

Let N be the number of states in this Markov chain.

1. What is the stationary distribution?

2. How quickly does it converge?

Solution

1. Since every transition has a matching reverse transition with the same transition probability, the chain is reversible with a uniform stationary distribution $\pi_H = 1/N$.

2. Here's a coupling that coalesces in at most $4m/3 + 2$ expected steps:

   (a) If X_t and Y_t are both trees, then send them to the same tree with probability 1/2; else let them both add edges independently (or we could have them add the same edge—it doesn't make any difference to the final result).

   (b) If only one of X_t and Y_t is a tree, with probability 1/2 scramble the tree while attempting to remove an edge from the non-tree, and the rest of the time scramble the non-tree (which has no effect) while attempting to add an edge to the tree. Since the non-tree has at least three edges that can be removed, this puts $(X_{t+1}, Y_{t+1}$ in the two-tree case with probability at least $3/2m$.

   (c) If neither X_t nor Y_t is a tree, attempt to remove an edge from both. Let S and T be the sets of edges that we can remove from X_t and Y_t, respectively, and let $k = \min(|S|, |T|) \geq 3$. Choose k edges from each of S and T and match them, so that if we remove one edge from each pair, we also remove the other edge. As in the previous case, this puts $(X_{t+1}, Y_{t+1}$ in the two-tree case with probability at least $3/2m$.

   To show this coalesces, starting from an arbitrary state, we reach a two-tree state in at most $2m/3$ expected steps. After one more step, we either coalesce (with probability 1/2) or restart from a new arbitrary state. This gives an expected coupling time of at most $2(2m/3 + 1) = 4m/3 + 2$ as claimed.

A.6 Assignment 6: due Monday, 2011-04-25, at 17:00

A.6.1 Sparse satisfying assignments to DNFs

Given a formula in disjunctive normal form, we'd like to estimate the number of satisfying assignments in which exactly w of the variables are true. Give a fully polynomial-time randomized approximation scheme for this problem.

Solution

Essentially, we're going to do the Karp-Luby convering trick [KL85] described in Section 8.3, but will tweak the probability distribution when we generate our samples so that we only get samples with weight w.

Let U be the set of assignment with weight w (there are exactly $\binom{n}{w}$ such assignments, where n is the number of variables). For each clause C_i, let $U_i = \{x \in U | C_i(x) = 1\}$. Now observe that:

1. We can compute $|U_i|$. Let $k_i = |C_i|$ be the number of variables in C_i and $k_i^+ = \left|C_i^+\right|$ the number of variables that appear in positive form in C_i. Then $|U_i| = \binom{n-k_i}{w-k_i^+}$ is the number of ways to make a total of w variables true using the remaining $n - k_i$ variables.

2. We can sample uniformly from U_i, by sampling a set of $w - k_i^+$ true variables not in C_i uniformly from all variables not in C_i.

3. We can use the values computed for $|U_i|$ to sample i proportionally to the size of $|U_i|$.

So now we sample pairs (i, x) with $x \in U_i$ uniformly at random by sampling i first, then sampling $x \in U_i$. As in the original algorithm, we then count (i, x) if and only if C_i is the leftmost clause for which $C_i(x) = 1$. The same argument that at least $1/m$ of the (i, x) pairs count applies, and so we get the same bounds as in the original algorithm.

A.6.2 Detecting duplicates

Algorithm A.1 attempts to detect duplicate values in an input array S of length n.

It's easy to see that Algorithm A.1 never returns **true** unless some value appears twice in S. But maybe it misses some duplicates it should find.

```
 1 Initialize A[1...n] to ⊥
 2 Choose a hash function h
 3 for i ← 1...n do
 4     x ← S[i]
 5     if A[h(x)] = x then
 6         return true
 7     else
 8         A[h(x)] ← x
 9     end if
10 end for
11 return false
```

Algorithm A.1: Dubious duplicate detector

1. Suppose h is a random function. What is the worst-case probability that Algorithm A.1 returns **false** if S contains two copies of some value?

2. Is this worst-case probability affected if h is drawn instead from a 2-universal family of hash functions?

Solution

1. Suppose that $S[i] = S[j] = x$ for $i < j$. Then the algorithm will see x in $A[h(x)]$ on iteration j and return **true**, unless it is overwritten by some value $S[k]$ with $i < k < j$. This occurs if $h(S[k]) = h(x)$, which occurs with probability exactly $1-(1-1/n)^{j-i-1}$ if we consider all possible k. This quantity is maximized at $1-(1-1/n)^{n-2} \approx 1-(1-1/n)^2/e \approx 1-(1-1/2n)/e$ when $i = 1$ and $j = n$.

2. As it happens, the algorithm can fail pretty badly if all we know is that h is 2-universal. What we can show is that the probability that some $S[k]$ with $i < j < k$ gets hashed to the same place as $x = S[i] = S[j]$ in the analysis above is at most $(j-i-1)/n \leq (n-2)/n = (1-2/n)$, since each $S[k]$ has at most a $1/n$ chance of colliding with x and the union bound applies. But it is possible to construct a 2-universal family for which we get exactly this probability in the worst case.

   Let $U = \{0 \ldots n\}$, and define for each a in $\{0 \ldots n-1\}$ $h_a : U \to n$ by $h_a(n) = 0$ and $h_a(x) = (x + a) \bmod n$ for $x \neq n$. Then $H = \{h_a\}$ is 2-universal, since if $x \neq y$ and neither x nor y is n, $\Pr[h_a(x) = h_a(y)] = 0$,

and if one of x or y is n, $\Pr[h_a(x) = h_a(y)] = 1/n$. But if we use this family in Algorithm A.1 with $S[1] = S[n] = n$ and $S[k] = k$ for $1 < k < n$, then there are $n-2$ choices of a that put one of the middle values in $A[0]$.

A.6.3 Balanced Bloom filters

A clever algorithmist decides to solve the problem of Bloom filters filling up with ones by capping the number of ones at $m/2$. As in a standard Bloom filter, an element is inserted by writing ones to $A[h_1(x)], A[h_2(x)], \ldots, A[h_k(x)]$; but after writing each one, if the number of one bits in the bit-vector is more than $m/2$, one of the ones in the vector (chosen uniformly at random) is changed back to a zero.

Because some of the ones associated with a particular element might be deleted, the membership test answers yes if at least $3/4$ of the bits $A[h_1(x)] \ldots A[h_k(x)]$ are ones.

To simplify the analysis, you may assume that the h_i are independent random functions. You may also assume that $(3/4)k$ is an integer.

1. Give an upper bound on the probability of a false positive when testing for a value x that has never been inserted.

2. Suppose that we insert x at some point, and then follow this insertion with a sequence of insertions of new, distinct values $y_1, y_2, \ldots$. Assuming a worst-case state before inserting x, give asymptotic upper and lower bounds on the expected number of insertions until a test for x fails.

Solution

1. The probability of a false positive is maximized when exactly half the bits in A are one. If x has never been inserted, each $A[h_i(x)]$ is equally likely to be zero or one. So $\Pr[\text{false positive for } x] = \Pr[S_k \geq (3/4)k]$ when S_k is a binomial random variable with parameters $1/2$ and k. Chernoff bounds give

$$\begin{aligned}\Pr[S_k \geq (3/4)k] &= \Pr[S_k \geq (3/2)\,\mathrm{E}[S_k]] \\ &\leq \left(\frac{e^{1/2}}{(3/2)^{3/2}}\right)^{k/2} \\ &\leq (0.94734)^k.\end{aligned}$$

We can make this less than any fixed ϵ by setting $k \geq 20\ln(1/\epsilon)$ or thereabouts.

2. For false negatives, we need to look at how quickly the bits for x are eroded away. A minor complication is that the erosion may start even as we are setting $A[h_1(x)] \ldots A[h_k(x)]$.

   Let's consider a single bit $A[i]$ and look at how it changes after (a) setting $A[i] = 1$, and (b) setting some random $A[r] = 1$.

   In the first case, $A[i]$ will be 1 after the assignment unless it is set back to zero, which occurs with probability $\frac{1}{m/2+1}$. This distribution does not depend on the prior value of $A[i]$.

   In the second case, if $A[i]$ was previously 0, it becomes 1 with probability

$$\begin{aligned}\frac{1}{m}\left(1 - \frac{1}{m/2+1}\right) &= \frac{1}{m} \cdot \frac{m/2}{m/2+1} \\ &= \frac{1}{m+2}.\end{aligned}$$

   If it was previously 1, it becomes 0 with probability

$$\frac{1}{2} \cdot \frac{1}{m/2+1} = \frac{1}{m+2}.$$

   So after the initial assignment, $A[i]$ just flips its value with probability $\frac{1}{m+2}$.

   It is convenient to represent $A[i]$ as ± 1; let $X_i^t = -1$ if $A[i] = 0$ at time t, and 1 otherwise. Then X_i^t satisfies the recurrence

$$\begin{aligned}\mathrm{E}[X_i^{t+1} | X_i^t] &= \frac{m+1}{m+2} X_i^t - \frac{1}{m+2} X_i^t \\ &= \frac{m}{m+2} X_i^t. \\ &= \left(1 - \frac{2}{m+2}\right) X_i^t.\end{aligned}$$

   We can extend this to $\mathrm{E}[X_i^t | X_i^0] = \left(1 - \frac{2}{m+2}\right)^t X_i^0 \approx e^{-2t/(m+2)} X_i^0$.

   Similarly, after setting $A[i] = 1$, we get $\mathrm{E}[X_i] = 1 - 2\frac{1}{m/2+1} = 1 - \frac{4}{2m+1} = 1 - o(1)$.

Let $S^t = \sum_{i=1}^{k} X^t_{h_i(x)}$. Let 0 be the time at which we finish inserting x. Then each for each i we have

$$1 - o(1)e^{-2k/(m+2)} \leq \mathrm{E}[X^0_{h_i(x)}] \leq 1 - o(1),$$

from which it follows that

$$k(1 - o(1))e^{-2k/(m+2)} \leq \mathrm{E}[S^0] \leq 1 - o(1)$$

and in general that

$$k(1 - o(1))e^{-2(k+t)/(m+2)} \leq \mathrm{E}[S^t] \leq 1 - o(1)e^{-2t/(m+2)}.$$

So for any fixed $0 < \epsilon < 1$ and sufficiently large m, we will have $\mathrm{E}[S^t] = \epsilon k$ for some t' where $t \leq t' \leq k + t$ and $t = \Theta(m \ln(1/\epsilon))$.

We are now looking for the time at which S^t drops below $k/2$ (the $k/2$ is because we are working with ± 1 variables). We will bound when this time occurs using Markov's inequality.

Let's look for the largest time t with $\mathrm{E}[S^t] \geq (3/4)k$. Then $\mathrm{E}[k - S^t] \leq k/4$ and $\Pr[k - S^t \geq k/2] \leq 1/2$. It follows that after $\Theta(m) - k$ operations, x is still visible with probability $1/2$, which implies that the expected time at which it stops being visible is at least $(\Omega(m) - k)/2$. To get the expected number of insert operations, we divide by k, to get $\Omega(m/k)$.

For the upper bound, apply the same reasoning to the first time at which $\mathrm{E}[S^t] \leq k/4$. This occurs at time $O(m)$ at the latest (with a different constant), so after $O(m)$ steps there is at most a $1/2$ probability that $S^t \geq k/2$. If S^t is still greater than $k/2$ at this point, try again using the same analysis; this gives us the usual geometric series argument that $\mathrm{E}[t] = O(m)$. Again, we have to divide by k to get the number of insert operations, so we get $O(m/k)$ in this case.

Combining these bounds, we have that x disappears after $\Theta(m/k)$ insertions on average. This seems like about what we would expect.

A.7 Final exam

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are four problems on this exam, each worth 20 points, for a total of 80 points. You have approximately three hours to complete this exam.

A.7.1 Leader election

Suppose we have n processes and we want to elect a leader. At each round, each process flips a coin, and drops out if the coin comes up tails. We win if in some round there is exactly one process left.

Let $T(n)$ be the probability that this event eventually occurs starting with n processes. For small n, we have $T(0) = 0$ and $T(1) = 1$. Show that there is a constant $c > 0$ such that $T(n) \geq c$ for all $n > 1$.

Solution

Let's suppose that there is some such c. We will necessarily have $c \leq 1 = T(1)$, so the induction hypothesis will hold in the base case $n = 1$.

For $n \geq 2$, compute

$$\begin{aligned} T(n) &= \sum_{k=0}^{n} 2^{-n} \binom{n}{k} T(k) \\ &= 2^{-n}T(n) + 2^{-n}nT(1) + \sum_{k=2}^{n-1} 2^{-n} \binom{n}{k} T(k) \\ &\geq 2^{-n}T(n) + 2^{-n}n + 2^{-n}(2^n - n - 2)c. \end{aligned}$$

Solve for $T(n)$ to get

$$\begin{aligned} T(n) &\geq \frac{n + (2^n - n - 2)c}{2^n - 1} \\ &= c\left(\frac{2^n - n - 2 + n/c}{2^n - 1}\right). \end{aligned}$$

This will be greater than or equal to c if $2^n - n - 2 + n/c \geq 2^n - 1$ or $n/c \geq n + 1$, which holds if $c \leq \frac{n}{n+1}$. The worst case is $n = 2$ giving $c = 2/3$.

Valiant and Vazirani [VV86] used this approach to reduce solving general instances of SAT to solving instances of SAT with unique solutions; they prove essentially the result given above (which shows that fixing variables in a SAT formula is likely to produce a SAT formula with a unique solution at some point) with a slightly worse constant.

A.7.2 Two-coloring an even cycle

Here is a not-very-efficient algorithm for 2-coloring an even cycle. Every node starts out red or blue. At each step, we pick one of the n nodes

uniformly at random, and change its color if it same color as at least one of its neighbors. We continue until no node has the same color as either of its neighbors.

Suppose that in the initial state there are exactly two monochromatic edges. What is the worst-case expected number of steps until there are no monochromatic edges?

Solution

Suppose we have a monochromatic edge surrounded by non-monochrome edges, e.g. $RBRRBR$. If we pick one of the endpoints of the edge (say the left endpoint in this case), then the monochromatic edge shifts in the direction of that endpoint: $RBBRBRB$. Picking any node not incident to a monochromatic edge has no effect, so in this case there is no way to increase the number of monochromatic edges.

It may also be that we have two adjacent monochromatic edges: $BRRRB$. Now if we happen to pick the node in the middle, we end up with no monochromatic edges ($BRBRB$) and the process terminates. If on the other hand we pick one of the nodes on the outside, then the monochromatic edges move away from each other.

We can thus model the process with 2 monochromatic edges as a random walk, where the difference between the leftmost nodes of the edges (mod n) increases or decreases with equal probability $2/n$ except if the distance is 1 or -1; in this last case, the distance increases (going to 2 or -2) with probability $2/n$, but decreases only with probability $1/n$. We want to know when this process hits 0 (or n).

Imagine a random walk starting from position k with absorbing barriers at 1 and $n-1$. This reaches 1 or $n-1$ after $(k-1)(n-1-k)$ steps on average, which translates into $(n/4)(k-1)(n-1-k)$ steps of our original process if we take into account that we only move with probability $4/n$ per time unit. This time is maximized by setting $k = n/2$, which gives $(n/4)(n/2-1)^2 = n^3/16 - n^2/4 + n/4$ expected time units to reach 1 or $n-1$ for the first time.

At 1 or $n-1$, we wait an addition $n/3$ steps on average; then with probability $1/3$ the process finishes and with probability $2/3$ we start over from position 2 or $n-2$; in the latter case, we run $(n/4)(n-3) + n/3$ time units on average before we may finish again. On average, it takes 3 attempts to finish, so this latter phase of the process adds $(3/4)n(n-3)+n = (3/4)n^2 - (5/4)n$ steps.

Adding up all of the costs gives $n^3/16 - n^2/4 + n/4 + n/3 + (3/4)n^2 -$

$(5/4)n = \frac{1}{16}n^3 + \frac{1}{2}n^2 - \frac{2}{3}n$ steps.

A.7.3 Finding the maximum

```
1 Randomly permute A
2 m ← −∞
3 for i ← 1 . . . n do
4     if A[i] > m then
5         m ← A[i]
6     end if
7 end for
8 return m
```

Algorithm A.2: Randomized max-finding algorithm

Suppose that we run Algorithm A.2 on an array with n elements, all of which are distinct. What is the expected number of times Line 5 is executed as a function of n?

Solution

Let X_i be the indicator variable for the event that Line 5 is executed on the i-th pass through the loop. This will occur if $A[i]$ is greater than $A[j]$ for all $j < i$, which occurs with probability exactly $1/i$ (given that A has been permuted randomly). So the expected number of calls to Line 5 is $\sum i = 1^n \mathrm{E}[X_i] = \sum_{i=1}^{n} \frac{1}{i} = H_n$.

A.7.4 Random graph coloring

Let G be a random d-regular graph on n vertices, that is, a graph drawn uniformly from the family of all n-vertex graph in which each vertex has exactly d neighbors. Color the vertices of G red or blue independently at random.

1. What is the expected number of monochromatic edges in G?
2. Show that the actual number of monochromatic edges is tightly concentrated around its expectation.

Solution

The fact that G is itself a random graph is a red herring; all we really need to know is that it's d-regular.

1. Because G has exactly $dn/2$ edges, and each edge has probability $1/2$ of being monochromatic, the expected number of monochromatic edges is $dn/4$.

2. This is a job for Azuma's inequality. Consider the vertex exposure martingale. Changing the color of any one vertex changes the number of monochromatic edges by at most d. So we have $\Pr[|X - \mathrm{E}[X]| \geq t] \leq 2\exp\left(-t^2/2\sum_{i=1}^{n} d^2\right) = 2e^{-t^2/2nd^2}$, which tells us that the deviation is likely to be not much more than $O(d\sqrt{n})$.

Appendix B

Sample assignments from Spring 2009 semester

B.1 Final exam, Spring 2009

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are four problems on this exam, each worth 20 points, for a total of 80 points. You have approximately three hours to complete this exam.

B.1.1 Randomized mergesort (20 points)

Consider the following randomized version of the mergesort algorithm. We take an unsorted list of n elements and split it into two lists by flipping an independent fair coin for each element to decide which list to put it in. We then recursively sort the two lists, and merge the resulting sorted lists. The merge procedure involves repeatedly comparing the smallest element in each of the two lists and removing the smaller element found, until one of the lists is empty.

Compute the expected number of comparisons needed to perform this final merge. (You do not need to consider the cost of performing the recursive sorts.)

Solution

Color the elements in the final merged list red or blue based on which sublist they came from. The only elements that do not require a comparison to insert into the main list are those that are followed only by elements of the

same color; the expected number of such elements is equal to the expected length of the longest monochromatic suffix. By symmetry, this is the same as the expected longest monochromatic prefix, which is equal to the expected length of the longest sequence of identical coin-flips.

The probability of getting k identical coin-flips in a row followed by a different coin-flip is exactly 2^{-k}; the first coin-flip sets the color, the next $k-1$ must follow it (giving a factor of 2^{-k+1}, and the last must be the opposite color (giving an additional factor of 2^{-1}). For n identical coin-flips, there is a probability of 2^{-n+1}, since we don't need an extra coin-flip of the opposite color. So the expected length is $\sum_{k=1}^{n-1} k2^{-k} + n2^{-n+1} = \sum_{k=0}^{n} k2^{-k} + n2^{-n}$.

We can simplify the sum using generating functions. The sum $\sum_{k=0}^{n} 2^{-k}z^k$ is given by $\frac{1-(z/2)^{n+1}}{1-z/2}$. Taking the derivative with respect to z gives $\sum_{k=0}^{n} 2^{-k}kz^{k-1} = (1/2)\frac{1-(z/2)^{n+1}}{1-z/2}^2 + (1/2)\frac{(n+1)(z/2)^n}{1-z/2}$. At $z = 1$ this is $2(1 - 2^{-n-1}) - 2(n + 1)2^{-n} = 2 - (n+2)2^{-n}$. Adding the second term gives $\mathrm{E}[X] = 2 - 2 \cdot 2^{-n} = 2 - 2^{-n+1}$.

Note that this counts the expected number of elements for which we do not have to do a comparison; with n elements total, this leaves $n-2+2^{-n+1}$ comparisons on average.

B.1.2 A search problem (20 points)

Suppose you are searching a space by generating new instances of some problem from old ones. Each instance is either good or bad; if you generate a new instance from a good instance, the new instance is also good, and if you generate a new instance from a bad instance, the new instance is also bad.

Suppose that your start with X_0 good instances and Y_0 bad instances, and that at each step you choose one of the instances you already have uniformly at random to generate a new instance. What is the expected number of good instances you have after n steps?

Hint: Consider the sequence of values $\{X_t/(X_t + Y_t)\}$.

Solution

We can show that the suggested sequence is a martingale, by computing

$$\begin{aligned}
\mathrm{E}\left[\frac{X_{t+1}}{X_{t+1}Y_{t+1}}|X_t, Y_t\right] &= \frac{X_t}{X_t+Y_t}\frac{X_t+1}{X_t+Y_t+1} + \frac{Y_t}{X_t+Y_t}\frac{X_t}{X_t+Y_t+1} \\
&= \frac{X_t(X_t+1)Y_tX_t}{(X_t+Y_t)+(X_t+Y_t+1)} \\
&= \frac{X_t(X_t+Y_t+1)}{(X_t+Y_t)+(X_t+Y_t+1)} \\
&= \frac{X_t}{X_t+Y+t}.
\end{aligned}$$

From the martingale property we have $\mathrm{E}\left[\frac{X_n}{X_n+Y_n}\right] = \frac{X_0}{X_0+Y+0}$. But $X_n + Y_n = X_0 + Y_0 + n$, a constant, so we can multiply both sides by this value to get $\mathrm{E}[X_n] = X_0\left(\frac{X_0+Y_0+n}{X_0+Y_0}\right)$.

B.1.3 Support your local police (20 points)

At one point I lived in a city whose local police department supported themselves in part by collecting fines for speeding tickets. A speeding ticket would cost 1 unit (approximately $100), and it was unpredictable how often one would get a speeding ticket. For a price of 2 units, it was possible to purchase a metal placard to go on your vehicle identifying yourself as a supporter of the police union, which (at least according to local legend) would eliminate any fines for subsequent speeding tickets, but which would not eliminate the cost of any previous speeding tickets.

Let us consider the question of when to purchase a placard as a problem in on-line algorithms. It is possible to achieve a strict[1] competitive ratio of 2 by purchasing a placard after the second ticket. If one receives fewer than 2 tickets, both the on-line and off-line algorithms pay the same amount, and at 2 or more tickets the on-line algorithm pays 4 while the off-line pays 2 (the off-line algorithm purchased the placard before receiving any tickets at all).

1. Show that no deterministic algorithm can achieve a lower (strict) competitive ratio.

2. Show that a randomized algorithm can do so, against an oblivious adversary.

[1] I.e., with no additive constant.

Solution

1. Any deterministic algorithm essentially just chooses some fixed number m of tickets to collect before buying the placard. Let n be the actual number of tickets issued. For $m = 0$, the competitive ratio is infinite when $n = 0$. For $m = 1$, the competitive ratio is 3 when $n = 1$. For $m > 2$, the competitive ratio is $(m + 2)/2 > 2$ when $n = m$. So $m = 2$ is the optimal choice.

2. Consider the following algorithm: with probability p, we purchase a placard after 1 ticket, and with probability $q = 1 - p$, we purchase a placard after 2 tickets. This gives a competitive ratio of 1 for $n = 0$, $1 + 2p$ for $n = 1$, and $(3p + 4q)/2 = (4 - p)/2 = 2 - p/2$ for $n \geq 2$. There is a clearly a trade-off between the two ratios $1+2p$ and $2-p/2$. The break-even point is when they are equal, at $p = 2/5$. This gives a competitive ratio of $1 + 2p = 9/5$, which is less than 2.

B.1.4 Overloaded machines (20 points)

Suppose n^2 jobs are assigned to n machines with each job choosing a machine independently and uniformly at random. Let the load on a machine be the number of jobs assigned to it. Show that for any fixed $\delta > 0$ and sufficiently large n, there is a constant $c < 1$ such that the maximum load exceeds $(1 + \delta)n$ with probability at most nc^n.

Solution

This is a job for Chernoff bounds. For any particular machine, the load S is a sum of independent indicator variables and the mean load is $\mu = n$. So we have

$$\Pr[S \geq (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^n .$$

Observe that $e^\delta/(1+\delta)^{1+\delta} < 1$ for $\delta > 0$. One proof of this fact is to take the log to get $\delta-(1+\delta)\log(1+\delta)$, which equals 0 at $\delta = 0$, and then show that the logarithm is decreasing by showing that $\frac{d}{d\delta} \cdots = 1 - \frac{1+\delta}{1+\delta} - \log(1 + \delta) = -\log(1 + \delta) < 0$ for all $\delta > 0$.

So we can let $c = e^\delta/(1 + \delta)^{1+\delta}$ to get a bound of c^n on the probability that any particular machine is overloaded and a bound of nc^n (from the union bound) on the probability that any of the machines is overloaded.

Appendix C

Discrete probability theory

In this appendix, we summarize the parts of **discrete probability theory** that we need for the course. This is not really a substitute for reading an actual probability theory book like Feller [Fel68] or Grimmett and Stirzaker [GS01].

C.1 Probability spaces and events

A **discrete probability space** is a countable set Ω of **points** or **outcomes** ω. Each ω in Ω has a **probability** $\Pr[\omega]$, which is a real value with $0 \leq \Pr[\omega] \leq 1$. It is required that $\sum_{\omega \in \Omega} = 1$.

An **event** A is a subset of Ω; its probability is $\Pr[A] = \sum_{\omega \in A} \Pr[\omega]$. We require that $\Pr[\Omega] = 1$, and it is immediate from the definition that $\Pr[\emptyset] = 0$.

The **complement** $\overline{A}$ or $\neg A$ of an event A is the event $\Omega - A$. It is always the case that $\Pr[\neg A] = 1 - \Pr[A]$.

This fact is a special case of the general principle that if $A_1, A_2, \ldots$ forms a **partition** of Ω—that is, if $A_i \cap A_j = \emptyset$ when $i \neq j$ and $\bigcup A_i = \Omega$—then $\sum \Pr[A_i] = 1$. It happens to be the case that $\neg A$ and A form a partition of Ω consisting of exactly two elements.

Whenever $A_1, A_2, \ldots$ are disjoint events (i.e., when $A_i \cap A_j = \emptyset$ for all $i \neq j$), it is the case that $\Pr\left[\bigcup A_i\right] = \sum \Pr[A_i]$. This fact does not hold in general for events that are not disjoint.

Any two events A and B satisfy the **inclusion-exclusion formula**: $\Pr[A] + \Pr[B] = \Pr[A \cup B] + \Pr[A \cap B]$. This can be used to calculate $\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$ if we know all the probabilities on the right-hand side.

All of these facts can be proven directly from the definition of probabilities for events.

More general probability spaces consist of a triple $(\Omega, \mathcal{F}, \Pr)$ where Ω is a set of points, $\mathcal{F}$ is a **σ-algebra** (a family of subsets of Ω that contains Ω and is closed under complement and countable unions) of **measurable sets**, and Pr is a function from $\mathcal{F}$ to $[0, 1]$ that gives $\Pr[\Omega] = 1$ and satisfies **countable additivity**: when $A_1, \ldots$ are disjoint, $\Pr[\bigcup A_i] = \sum \Pr[A_i]$. This definition is needed for uncountable spaces, because (under certain set-theoretic assumptions) we may not be able to assign a meaningful probability to all subsets of Ω.

C.1.1 Conditional probability and independence

The **probability of A conditioned on B** or **probability of A given B**, written $\Pr[A|B]$, is defined to be $\frac{\Pr[A \cap B]}{\Pr[B]}$, provided $\Pr[B \neq 0]$. (If $\Pr[B] = 0$, we can't condition on B.) Such conditional probabilities represent the effect of restricting our probability space to just B, which can think of as computing the probability of each event if we know that B occurs. The intersection in the numerator limits A to circumstances where B occurs, while the denominator normalizes the probabilities so that, for example, $\Pr[\Omega|B] = \Pr[B|B] = 1$.

We can use conditional probabilities to calculate $\Pr[A \cap B] = \Pr[B] \Pr[A|B] = \Pr[A] \Pr[B|A]$. In many cases, knowing that B occurs tells us nothing about whether A occurs; if so, we have $\Pr[A|B] = \Pr[B]$ and we say that A and B are **independent events**. The formal definition is that A and B are independent when $\Pr[A \cap B] = \Pr[A] \Pr[B]$; this is equivalent to the conditional-probability version when $\Pr[B] \neq 0$ but makes the symmetry more apparent and allows for the possibility that one or both events have zero probability.

A set of events $A_1, A_2, \ldots$ is independent if $\Pr[A_i|B] = \Pr[A_i]$ when B is any Boolean formula of the A_j for $j \neq i$. A set of events $A_1, A_2, \ldots$ is **pairwise independent** if each A_i and A_j, $i \neq j$ are independent. It is possible for a set of events to be pairwise independent but not independent; a simple example is when A_1 and A_2 are the events that two independent coins come up heads and A_3 is the event that both coins come up with the same value. The general version of pairwise independence is **k-wise independence**, which means that any subset of k (or fewer) events are independent.

C.1.2 Random variables

A **random variable** on a probability space Ω is just a function with domain Ω. Rather than writing a random variable as $f(\omega)$ everywhere, the convention is to write a random variable as a capital letter (X, Y, S, etc.) and make the argument implicit. Variables that aren't random (or aren't variable) are written in lowercase.

Random variables may be combined using standard arithmetic operators, have functions applied to them, etc., to get new random variables. For example, the random variable $X + Y$ is a function from Ω that takes on the value $X(\omega) + Y(\omega)$ on each point ω.

Events involving random variables are often written in square brackets, in what is called **Iverson notation**; for example, the event $[X = 2Y]$ is the set of all ω such that $X(\omega) = 2Y(\omega)$. Such events are themselves often interpreted as random variables, known as **indicator random variables**. An indicator random variable is 1 if the event occurs and 0 otherwise.

The **probability mass function** of a random variable gives $\Pr[X = x]$ for each possible value x. For two random variables, the **joint probability mass function** gives $\Pr[X = x \wedge Y = y]$ for each pair of values x and y (this generalizes in the obvious way for more than two variables). We will often refer to the probability mass function as giving the **distribution** or **joint distribution** of a random variable or collection of random variables, even though distribution technically means the function $F(x) = \Pr[X \leq x]$, which is generally not directly computable from the probability mass function for **continuous random variables** that take on uncountably many values. (To the extent that we can, we will try to avoid continuous random variables, and the rather messy integration theory needed to handle them.)

Two or more random variables are **independent** if all sets of events involving different random variables are independent.

C.1.3 Expectation and moments

The **expectation** or **expected value** of a random variable X is given by $\mathrm{E}[X] = \sum_x x \Pr[X = x]$. This is essentially an average value of X weighted by probability, and it only makes sense if X takes on values that can be summed in this way (e.g., real or complex values, or vectors in a real- or complex-valued vector space). Even if the expectation makes sense, it may be that a particular random variable X doesn't have an expectation, because the sum fails to converge.

For variables that have expectations, the expectation operator has sev-

eral very nice properties. Chief among these is **linearity of expectation**: $\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y]$ for all random variables X and Y (in general, $\mathrm{E}\left[\sum X_i\right] = \sum \mathrm{E}[X_i]$) and $\mathrm{E}[cX] = c\,\mathrm{E}[X]$ when c is a constant. This property follows immediately from the definition of expectation.

When two random variables X and Y are independent, it also holds that $\mathrm{E}[XY] = \mathrm{E}[X]\,\mathrm{E}[Y]$. This is not true for arbitrary random variables.

The k-th **moment** of a random variable is $\mathrm{E}[X^k]$. Often we consider the k-th **central moment** instead, defined as $\mathrm{E}[(X - \mathrm{E}[X])^k]$. The first central moment is always zero. The second central moment $\mathrm{E}[(X - \mathrm{E}[X])^2]$ is known as the **variance** of X and is written as $\mathrm{Var}[X]$; it is a measure of how far X deviates from its expectation. It is possible to rewrite $\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2] = \mathrm{E}[X^2 - 2X\,\mathrm{E}[X] + \mathrm{E}[X]^2] = \mathrm{E}[X^2] - 2\,\mathrm{E}[X]^2 + \mathrm{E}[X]^2 = \mathrm{E}[X^2] - \mathrm{E}[X]^2$, which can simplify some calculations. Variance also has the useful property that $\mathrm{Var}[X + Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]$ when X and Y are independent.

Higher-order moments are used less often in analysis of randomized algorithms but have occasional applications.

C.1.4 Conditional expectation

We can also define a notion of **conditional expectation**, analogous to conditional probability. There are three versions of this, depending on how fancy we want to get about specifying what information we are conditioning on:

- The conditional expectation of X conditioned on an *event* A is written $\mathrm{E}[X|A]$ and defined by $\mathrm{E}[X|A] = \sum_x x \Pr[X = x|A] = \sum_x \frac{\Pr[X=x \wedge A]}{\Pr[A]}$. This is essentially the weighted average value of X if we know that A occurs.

- The expectation of X conditioned on a *random variable* Y, written $\mathrm{E}[X|Y]$, is a random variable. Formally, for each $\omega \in \Omega$, $\mathrm{E}[X|Y](\omega) = \mathrm{E}[X|Y = Y(\omega)]$. The intuition behind this definition is that $\mathrm{E}[X|Y]$ is a weighted-average estimate of X given that we know the value of Y but nothing else. Similar, we can define $\mathrm{E}[X|Y_1, Y_2, \ldots]$ to be the expected value of X given that we know the values of $Y_1, Y_2, \ldots$.

- The preceding is actually a special case of the expectation of X conditioned on a σ-algebra $\mathcal{F}$. Recall that a σ-algebra is a family of subsets of Ω that includes Ω and is closed under complement and countable union; for discrete probability spaces, this turns out to be the set of

all unions of equivalence classes for some equivalence relation on Ω,[1] and we think of $\mathcal{F}$ as representing knowledge of which equivalence class we are in, but not which point in the equivalence class we land on. An example would be if Ω consists of all values (X_1, X_2) obtained from two die rolls, and $\mathcal{F}$ consists of all sets A such that whenever one point ω with $X_1(\omega) + X_2(\omega) = s$ is in A, so is every other point ω' with $X_1(\omega') + X_2(\omega') = s$. (This is the σ-algebra **generated by** the random variable $X_1 + X_2$.)

A discrete random variable X is **measurable** $\mathcal{F}$ if every event $[X = x]$ is contained in $\mathcal{F}$; in other words, knowing only where we are in $\mathcal{F}$, we can compute exactly the value of X.

If X is not measurable $\mathcal{F}$, the best approximation we can make to it given that we only know where we are in $\mathcal{F}$ is $\mathrm{E}[X|\mathcal{F}]$, which is defined as a random variable Q that is (a) measurable $\mathcal{F}$; and (b) satisfies $\mathrm{E}[Q|A] = \mathrm{E}[X|A]$ for any non-null $A \in \mathcal{F}$. For discrete probability spaces, this just means that we replace X with its average value across each equivalence class: property (a) is satisfied because $\mathrm{E}[X|\mathcal{F}]$ is constant across each equivalence class, meaning that $[\mathrm{E}[X|\mathcal{F}] = x]$ is a union of equivalence classes, and property (b) is satisfied because we define $\mathrm{E}[\mathrm{E}[X|\mathcal{F}]|A = \mathrm{E}[X|A]$ for each equivalence class A, and the same holds for unions of equivalence classes by a simple calculation.

Most properties of expectations also hold for conditional expectations. For example, $\mathrm{E}[X+Y|Z] = \mathrm{E}[X|Z]+\mathrm{E}[Y|Z]$ and $\mathrm{E}[cX|Y] = c\,\mathrm{E}[X|Y]$ when c is constant. But there are some additional properties that are often useful.

One is that computing the expectation of a condition expectation yields the expectation: $\mathrm{E}[\mathrm{E}[X|Y]] = \mathrm{E}[X]$. This is often used in reverse, to introduce a conditional expectation and let us compute $\mathrm{E}[X]$ piecewise using the **law of total probability** $\mathrm{E}[X] = \mathrm{E}[X|Y] = \sum_y \mathrm{E}[X|Y = y]\Pr[Y = y]$. We can also do partial conditioning: $\mathrm{E}[\mathrm{E}[X|Y, Z]|Y] = \mathrm{E}[X|Y]$.

A useful generalization of linearity of expectation is that when Z is a function of Y, then $\mathrm{E}[X\,Z|Y] = Z\,\mathrm{E}[X|Y]$. Essentially, Z acts like a constant over each event $[Y = y]$, so we can pull it out.

[1] Proof: Let $\mathcal{F}$ be a σ-algebra over a countable set Ω. Let $\omega \approx \omega'$ if, for all A in $\mathcal{F}$, $\omega \in A$ if and only if $\omega' \in A$; this is an equivalence relation on Ω. To show that the equivalence classes of $\approx$ are elements of $\mathcal{F}$, for each $\omega'' \not\approx \omega$, let $A_{\omega''}$ be some element of $\mathcal{F}$ that contains ω but not ω''. Then $\bigcap_{\omega''} A_{\omega''}$ (a countable intersection of elements of $\mathcal{F}$) contains ω and all points $\omega' \approx \omega$ but no points $\omega'' \not\approx \omega$; in other words, it's the equivalence class of ω. Since there are only countably many such equivalence classes, we can construct all the elements of $\mathcal{F}$ by taking all possible unions of them.

Bibliography

[ABMRT96] Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup. Static dictionaries on AC^0 RAMs: Query time $\theta(\sqrt{\log n / \log \log n})$ is necessary and sufficient. In *FOCS*, pages 441–450, 1996.

[Abr88] Karl Abrahamson. On achieving consensus using a shared memory. In *Proceedings of the 2nd annual ACM-SIAM symposium on Discrete algorithms*, pages 291–302, 1988.

[AC08] Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. *Journal of the ACM*, 55(5):20, October 2008.

[ACH10] Hagit Attiya and Keren Censor-Hillel. Lower bounds for randomized consensus under a weak adversary. *SIAM J. Comput.*, 39(8):3885–3904, 2010.

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 75–83, Washington, DC, USA, 1978. IEEE Computer Society.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. An O(n log n) sorting network. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 1–9, New York, NY, USA, 1983. ACM.

[Alo91] Noga Alon. A parallel algorithmic version of the local lemma. *Random Structures & Algorithms*, 2(4):367–378, 1991.

[AS92] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. John Wiley & Sons, 1992.

[Asp10] James Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. In *Proceedings of the Twenty-Ninth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 460–467, July 2010.

[AW96] James Aspnes and Orli Waarts. Randomized consensus in $o(n \log^2 n)$ operations per processor. *SIAM Journal on Computing*, 25(5):1024–1044, October 1996.

[Azu67] Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tôhoku Mathematical Journal*, 19(3):357–367, 1967.

[Bab79] László Babai. Monte-carlo algorithms in graph isomorphism testing. Technical Report D.M.S. 79-10, Université de Montréal, 1979.

[BD92] Dave Bayer and Persi Diaconis. Trailing the dovetail shuffle to its lair. *Annals of Applied Probability*, 2(2):294–313, 1992.

[Bec91] József Beck. An algorithmic approach to the lovász local lemma. i. *Random Structures & Algorithms*, 2(4):343–365, 1991.

[Bel57] Richard Earnest Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[Bol01] Béla Bollobás. *Random Graphs*. Cambridge University Press, second edition, 2001.

[Bro86] Andrei Z. Broder. How hard is to marry at random? (on the approximation of the permanent). In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, 28-30 May 1986, Berkeley, California, USA*, pages 50–58, 1986.

[Bro88] Andrei Z. Broder. Errata to "how hard is to marry at random? (on the approximation of the permanent)". In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 2-4 May 1988, Chicago, Illinois, USA*, page 551, 1988.

[BRSW06] Boaz Barak, Anup Rao, Ronen Shaltiel, and Avi Wigderson. 2-source dispersers for sub-polynomial entropy and ramsey graphs beating the frankl-wilson construction. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 671–680, New York, NY, USA, 2006. ACM.

[CCFC04] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[CIL94] Benny Chor, Amos Israeli, and Ming Li. Wait-free consensus using asynchronous hardware. *SIAM J. Comput.*, 23(4):701–712, 1994.

[CM03] Saar Cohen and Yossi Matias. Spectral bloom filters. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 241–252, 2003.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[CS00] Artur Czumaj and Christian Scheideler. Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász local lemma. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 30–39, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[CW77] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC '77, pages 106–112, New York, NY, USA, 1977. ACM.

[DG00] Martin Dyer and Catherine Greenhill. On markov chains for independent sets. *J. Algorithms*, 35:17–49, April 2000.

[DGM02] Martin Dyer, Catherine Greenhill, and Mike Molloy. Very rapid mixing of the glauber dynamics for proper colorings on

bounded-degree graphs. *Random Struct. Algorithms*, 20:98–114, January 2002.

[DHKP97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25(1):19–51, 1997.

[Dum56] A. I. Dumey. Indexing for rapid random-access memory. *Computers and Automation*, 5(12):6–9, 1956.

[Dye03] Martin Dyer. Approximate counting by dynamic programming. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 693–699, New York, NY, USA, 2003. ACM.

[EL75] Paul Erdős and Laszlo Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal, R. Rado, and V. T. Sós, editors, *Infinite and Finite Sets (to Paul Erdős on his 60th birthday)*, pages 609–627. North-Holland, 1975.

[Erd45] P. Erdős. On a lemma of Littlewood and Offord. *Bulletin of the American Mathematical Society*, 51(12):898–902, 1945.

[ES35] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[FCAB00] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, third edition, 1968.

[Fel71] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley, second edition, 1971.

[FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a Sparse Table with O(1) Worst Case Access Time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[GS01] Geoffrey R. Grimmett and David R. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001.

[Gur00] Venkatesen Guruswami. Rapidly mixing markov chains: A comparison of techniques. Available at `ftp://theory.lcs.mit.edu/pub/people/venkat/markov-survey.ps`, 2000.

[GW94] Michel X. Goemans and David P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM J. Discret. Math.*, 7:656–666, November 1994.

[Has70] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[HH80] P. Hall and C.C. Heyde. *Martingale Limit Theory and Its Application*. Academic Press, 1980.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[Hoa61a] C. A. R. Hoare. Algorithm 64: Quicksort. *Commun. ACM*, 4:321, July 1961.

[Hoa61b] C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4:321–322, July 1961.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[HST08] Maurice Herlihy, Nir Shavit, and Moran Tzafrir. Hopscotch hashing. In *Proceedings of the 22nd international symposium on Distributed Computing*, DISC '08, pages 350–364, Berlin, Heidelberg, 2008. Springer-Verlag.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[JLR00] Svante Janson, Tomasz Łuczak, and Andrezej Ruciński. *Random Graphs*. John Wiley & Sons, 2000.

[JS89] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

[Kar93] David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 21–30, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[KL85] Richard M. Karp and Michael Luby. Monte-carlo algorithms for the planar multiterminal network reliability problem. *J. Complexity*, 1(1):45–64, 1985.

[KM08] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better bloom filter. *Random Struct. Algorithms*, 33(2):187–218, 2008.

[KR99] V. S. Anil Kumar and H. Ramesh. Markovian coupling vs. conductance for the jerrum-sinclair chain. In *FOCS*, pages 241–252, 1999.

[KS10] Victor Korolev and Irina Shevtsova. An improvement of the Berry-Esseen inequality with applications to Poisson and mixed Poisson random sums. *Scandinavian Actuarial Journal*, 2010. In press. Available as `http://arxiv.org/abs/0912.2795`.

[KUW88] Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel search. *Journal of Computer and System Sciences*, 36(2):225–253, 1988.

[LAA87] Michael C. Loui and Hosame H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, pages 163–183, 1987.

[Li80] Shuo-Yen Robert Li. A martingale approach to the study of occurrence of sequence patterns in repeated experiments. *Annals of Probability*, 8(6):1171–1176, 1980.

[Lin92] Torgny Lindvall. *Lectures on the Coupling Method*. Wiley, 1992.

[Lub85] Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1985. ACM.

[LV97] Michael Luby and Eric Vigoda. Approximately counting up to four (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 682–687, New York, NY, USA, 1997. ACM.

[LV99] Michael Luby and Eric Vigoda. Fast convergence of the glauber dynamics for sampling independent sets. *Random Structures & Algorithms*, 15(3–4):229–241, 1999.

[McD89] Colin McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics, 1989: Invited Papers at the Twelfth British Combinatorial Conference*, pages 148–188, 1989.

[MH92] Colin McDiarmid and Ryan Hayward. Strong concentration for quicksort. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, SODA '92, pages 414–421, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[ML86] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi, editors, *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*, pages 149–159. Morgan Kaufmann, 1986.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, January 1998.

[Mos09] Robin A. Moser. A constructive proof of the Lovász local lemma. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing*, STOC '09, pages 343–350, New York, NY, USA, 2009. ACM.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MR98] Michael Molloy and Bruce Reed. Further algorithmic aspects of the local lemma. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 524–529, New York, NY, USA, 1998. ACM.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.

[MT10] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. *J. ACM*, 57:11:1–11:15, February 2010.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[MWW07] Elchanan Mossel, Dror Weitz, and Nicholas Wormald. On the hardness of sampling independent sets beyond the tree threshold. Available as `http://arxiv.org/abs/math/0701471`, 2007.

[Pag01] Rasmus Pagh. On the cell probe complexity of membership and perfect hashing. In *STOC*, pages 425–432, 2001.

[Pag06] Rasmus Pagh. Cuckoo hashing for undergraduates. Available at `http://www.it-c.dk/people/pagh/papers/cuckoo-undergrad.pdf`, 2006.

[PPR05] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal bloom filter replacement. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 823–829. SIAM, 2005.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[Pug90] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *J. Comput. Syst. Sci.*, 37:130–143, October 1988.

[Rou01] Salvador Roura. Improved master theorems for divide-and-conquer recurrences. *J. ACM*, 48:170–205, March 2001.

[RT87] Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, December 1987.

[Sri08] Aravind Srinivasan. Improved algorithmic versions of the Lovász local lemma. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 611–620, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[SS87] E. Shamir and J. Spencer. Sharp concentration of the chromatic number on random graphs gn,p. *Combinatorica*, 7:121–129, January 1987.

[Tod91] Seinosuke Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Val82] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, 1982.

[VB81] Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *STOC*, pages 263–277. ACM, 1981.

[VV86] Leslie G. Valiant and Vijay V. Vazirani. Np is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

[Wil04] David Bruce Wilson. Mixing times of lozenge tiling and card shuffling Markov chains. *Annals of Applied Probability*, 14(1):274–325, 2004.

Index

(r_1, r_2, p_1, p_2)-sensitive, 135
3-way cut, 159
$G(n, p)$, 42
ϵ-NNS, 135
ϵ-PLEB, 135
ϵ-nearest neighbor search, 135
ϵ-point localation in equal balls, 135
ϵ-PLEB, 135
σ-algebra, 183
σ-algebra, 185
    generated by a random variable, 186
k-CNF formula, 56
k-universal, 119
k-wise independence, 183
#DNF, 113, 114
#P, 112
#SAT, 112
0–1 random variable, 15
2-universal, 119

adapted, 74
adapted sequence of random variables, 69
adaptive adversary, 140
Adleman's theorem, 65
adversary, 140
    adaptive, 140
    oblivious, 140
advice, 65
agreement, 141
    probabilistic, 142
Aldous-Fill manuscript, 85
algorithm
    distributed, 140
annealing schedule, 97
anti-clique, 49
anti-concentration bound, 44
aperiodic, 78
approximation algorithm, 51
approximation ratio, 51
arithmetic coding, 149
augmenting path, 110
average-case analysis, 2
avoiding worst-case inputs, 9
Azuma's inequality, 33, 37
Azuma-Hoeffding inequality, 33

Bernoulli random variable, 15
Berry-Esseen theorem, 44
biased random walk, 73
bipartite graph, 110
bit-fixing routing, 32, 107
Bloom filter, 127, 171
    counting, 130
    spectral, 130
Bloomjoin, 129
branching process, 61
Byzantine processes, 140

canonical paths, 104
causal coupling, 99
central moment, 185
chaining, 118

Chapman-Kolmogorov equation, 77
Chebyshev's inequality, 23
Cheeger inequality, 103
chromatic number, 42
Chutes and Ladders, 12
clause, 50, 56, 114
clique, 49
closed set of states, 79
collision, 9, 118
common subsequence, 156
communicating states, 79
complement, 182
computation path, 65
computation tree, 65
concave function, 22
concentration bound, 23
conciliator, 142
conditional expectation, 3, 19, 185
conditional probability, 19, 183
conductance, 103
congestion, 106
consensus, 141
continuous random variable, 184
contraction, 6
convex function, 22
Cook reduction, 112
count-min sketch, 131
countable additivity, 183
countable Markov chain, 76
counting Bloom filter, 130
coupling, 80, 81
    causal, 99
    path, 92
coupling time, 87, 88
coupon collector problem, 12
cryptographic hash function, 10
cryptographically secure pseudorandom generator, 64
cuckoo hashing, 123
curse of dimensionality, 135
cut, 6

data stream computation, 131
dense, 110
dependency graph, 57
derandomization, 63
detailed balance equations, 82
dictionary, 118
discrete probability space, 182
discrete probability theory, 182
discretization, 138
disjunctive normal form, 114
distributed algorithm, 140
distributed algorithms, 10
distributed computing, 140
distribution, 184
DNF formula, 114
dominating set, 151
Doob martingale, 40, 66
dovetail shuffle, 91

Erdős-Szekeres theorem, 157
ergodic, 79
ergodic theorem, 80
event, 182
execution log, 60
expectation, 184
    conditional, 185
    conditioned on a σ-algebra, 185
    conditioned on a random variable, 185
    conditioned on an event, 185
expected time, 2
expected value, 1, 184
expected worst-case, 1
exponential generating function, 26

false positive, 127
filtration, 69
final exam, x
fingerprint, 10

fingerprinting, 10
finite Markov chain, 76
first passage time, 78
FPRAS, 112
fractional solution, 51, 52
fully polynomial randomized approximation scheme, 112
fully polynomial-time approximation scheme, 115

generated by, 186
generating function
    exponential, 26
Glauber dynamics, 92, 94
graph
    bipartite, 110
    random, 42

Hamming distance, 40
handwaving, 5, 60
hash function, 9, 118
    perfect, 122
hash functon
    universal, 119
hash table, 10, 118
hashing, 9
    cuckoo, 123
    hopscotch, 126
    locality-sensitive, 134
heavy hitter, 131
high-probability bound, 1
Hoare's FIND, 11
Hoeffding's inequality, 33
homogeneous, 76
hopscotch hashing, 126
hypercube network, 31, 41

inclusion-exclusion formula, 182
independent events, 183
independent random variables, 184
index, 118
indicator random variables, 184
indicator variable, 4
inequality
    Azuma's, 33, 37
    Azuma-Hoeffding, 33
    Chebyshev's, 23
    Cheeger, 103
    Hoeffding's, 33
    Markov's, 21
    McDiarmid's, 40
integer program, 51
intercommunicate, 79
irreducible, 79
Iverson notation, 184

joint distribution, 184
joint probability mass function, 184

Karger's min-cut algorithm, 5
Karp-Upfal-Wigderson bound, 12
key, 118
KNAPSACK, 115

Las Vegas algorithm, 7
law of total probability, 3, 186
leader election, 10
linear program, 51
linearity of expectation, 5, 185
Lipschitz function, 40, 98
literal, 56
Littlewood-Offord problem, 45
load balancing, 10
load factor, 119
locality-sensitive hashing, 134
Lovász Local Lemma, 54

Markov chain, 76
    reversible, 82
Markov process, 76
Markov's inequality, 21
martingale, 37, 69

Doob, 66
vertex exposure, 42
martingale difference sequence, 37
martingale property, 69
matching, 108
maximum cut, 48
McDiarmid's inequality, 40
mean recurrence time, 78
measurable, 186
measurable sets, 183
message-passing, 140
method of bounded differences, 40
method of conditional probabilities, 66
Metropolis, 83
Metropolis-Hastings
convergence, 97
Metropolis-Hastings algorithm, 83
min-cut, 6
minimum cut, 6
mixing, 87
mixing rate, 102
mixing time, 87
moment, 185
moment generating function, 26
Monte Carlo algorithm, 7
multigraph, 5

nearest neighbor search, 135
NNS, 135
non-null persistent, 78
non-uniform, 66
normal form
disjuctive, 114
null persistent, 78
number P, 112

objective function, 51
oblivious adversary, 140
open addressing, 118
optimal solution, 51
optional stopping theorem, 69
outcomes, 182

pairwise independence, 24
pairwise independent, 183
parallel computing, 140
partition, 182
path coupling, 91, 92
pending operation, 140
perfect hash function, 122
perfect matching, 110
perfect matchings, 113
period, 78
periodic, 78
permanent, 113
permutation routing, 31
Perron-Frobenius theorem, 80
persistent state, 78
pessimistic estimator, 67
pivot, 3
point location in equal balls, 135
point query, 131
points, 182
Poisson trial, 15
polynomial-time hierarchy, 113
preference, 141
probabilistic agreement, 142
probabilistic method, 10, 46
probabilistic recurrence, 42
probability, 182
conditional, 183
probability amplification, 63, 65
probability mass function, 184
probability of A conditioned on B, 183
probability of A given B, 183
probing, 119
pseudorandom generator, 63
cryptographically secure, 64

query
    point, 131
QuickSelect, 11
QuickSort, 2, 43

random graph, 42
random structure, 10
random variable, 1, 184
random walk, 72
    biased, 73
    unbiased, 72
    with one absorbing barrier, 73
    with two absorbing barriers, 72
randomized rounding, 51, 117
range coding, 149
rapid mixing, 87
reachable, 79
recurrence time, 78
reduction, 112
register, 140
rejection sampling, 83, 149
relax, 51
relaxation, 51
relaxation time, 102
renewal theorem, 80
reversible Markov chain, 82
ring-cover tree, 135
routing
    bit-fixing, 107
    permutation, 31
run, 154

sampling, 7, 9
    rejection, 83, 149
SAT, 174
satisfiability, 56
satisfiability problem, 50
satisfy, 51
seed, 63
separate chaining, 118
shared memory, 140
sharp P, 112
simulated annealing, 97
sketch, 131
    count-min, 131
sorting network, 47
spanning tree, 167
spectral Bloom filter, 130
spectral representation formula, 101
spectral theorem, 100
state space, 76
stationary distribution, 77, 80
stochastic matrix, 77
stochastic process, 76
stopping time, 69
strongly k-universal, 119
strongly 2-universal, 119
strongly connected, 79
supermartingale, 37, 42
symmetry breaking, 10

termination, 141
third moment, 45
time-reversed chain, 99
Toda's theorem, 113
total variation distance, 85
transient state, 78
transition probabilities, 76
tree
    ring-cover, 135
    witness, 60

unbiased random walk, 72
uniform, 66
universal hash family, 119

Valiant's theorem, 112
validity, 141
variance, 23, 185
vertex exposure martingale, 42

Wald's equation, 74
witness, 66
witness tree, 60
worst-case analysis, 2